Resource Management in Application Level Message Transfer Systems

By

BARRY JEFFREY BRACHMAN

B.Sc. Honours, The University of Regina, 1981
M.Sc., The University of British Columbia, 1983

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES
(Department of Computer Science)

We accept this thesis as conforming
to the required standard.

THE UNIVERSITY OF BRITISH COLUMBIA

June 1989

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of COMPUTER SCIENCE

The University of British Columbia
1956 Main Mall
Vancouver, Canada
V6T 1Y3

Date MAY 9, 1989

Abstract

This thesis is concerned with the design of the resource management components of application level, store-and-forward message transfer systems. Although these systems have for some time been used to transport electronic mail, there has been little investigation into designs that emphasize performance and correctness aspects. Current message transfer systems are loosely structured in that there is little, if any, end-to-end resource management.

The thesis begins by characterizing the message handling environment and comparing the message transfer approach to that of connection-based file transfer. Current message transfer systems have a fundamental limitation in that the largest message that can be transferred is determined by the amount of storage available at any of the intermediate hosts along the message's route. Major components of a message transfer system and design alternatives are discussed. Existing schemes that deal with solutions designed for lower networking levels are reviewed and shown to be inadequate in addressing the problems in the message handling environment.

A framework for designing message transfer systems is presented. Systems adhering to the design methodology address performance issues in a structured way. Two new techniques are central to this framework: message fragmentation and the message stream. Message fragmentation is introduced as a means of delivering arbitrary size messages. The message stream abstraction is the basis of flow control and congestion control. A hierarchical technique for deadlock prevention in the message handling environment is introduced. In this method, the structured buffer pool approach is used as a top level and is integrated with a second method at the bottom level to produce a practical, deadlock-free message transfer system. Methods for providing transit buffer management and recipient buffer space allocation are discussed. A simulation study of some of the performance aspects of message streams and recipient buffer space allocation is presented.

Contents

Abstract ii

List of Tables vii

List of Figures viii

Acknowledgements ix

1 Introduction 1

1.1 The Thesis 4

1.2 Overview of the Rest of the Thesis 6

2 Store-and-Forward Message Transfer 8

2.1 The Message Handling Environment 9

2.1.1 The X.400 Model 11

2.2 Characteristics of Message Transfer Systems 14

2.3 Current Message Transfer Systems 17

2.3.1 The EAN Message Handling System 18

2.3.2 Limitations of Current Message Transfer Systems 19

2.4 Connection-Based File Transfer Systems 20

2.4.1 Message Transfer *vs.* Connection-Based Transfer 21

2.5 Summary 26

3 Issues in the Design of Message Transfer Systems 28

3.1 Message Fragmentation 29

3.2 Buffer Management 31

3.2.1 Fragment Sizes . . . 32
3.2.1.1 Fixed Size Fragments . . . 32
3.2.1.2 Variable Size Fragments . . . 33
3.2.1.3 Hybrid . . . 33
3.2.2 Buffering Strategies . . . 34
3.2.2.1 Static Buffer Management . . . 35
3.2.2.2 Adaptive Buffer Management . . . 36
3.2.2.3 Buffer Management in the Message Handling Environment 37
3.2.3 Layering Concerns . . . 40
3.2.4 Reassembly and Gateways . . . 42
3.3 Deadlock . . . 43
3.4 Flow Control . . . 44
3.5 Congestion Control . . . 45
3.5.1 Network and Transport Level Congestion Control . . . 49
3.5.1.1 Buffer Class-Based Congestion Control . . . 52
3.5.2 Application Level Flow and Congestion Control . . . 53
3.6 Fairness . . . 56
3.7 Network Traffic Patterns . . . 62
3.8 Summary . . . 64

4 Store-and-Forward Deadlock Prevention **65**
4.1 Existing Techniques . . . 66
4.2 A Hierarchical Solution . . . 71
4.2.1 Intracluster Message Transport . . . 75
4.2.2 Basic Intracluster Message Transport . . . 76
4.2.2.1 The Connection-Based Method . . . 77
4.2.2.2 The Store-and-Forward Oriented Method . . . 78
4.2.3 Enhancements to Basic Intracluster Message Transport . . . 81
4.2.4 Variable Length Messages . . . 82
4.2.5 Application to Packet Switched and Datagram Networks . . . 84
4.3 Summary . . . 85

5 A Structure for Message Transfer Systems 86
5.1 Message Fragmentation 87
5.2 The Message Stream 90
5.2.1 Envelope Fragmentation 96
5.2.2 Message Stream Multiplexing 97
5.3 Congestion Control 99
5.4 Buffer Management 100
5.5 Recipient Buffer Space Allocation 105
5.5.1 Preallocation 106
5.5.2 Optimistic Transfer 109
5.5.3 Multirecipient Messages Having a Common Recipient MTA 110
5.5.4 A Recipient Buffer Management System 111
5.6 Fairness Issues 114
5.7 Tuning, Maintenance, and Disaster Recovery 115
5.8 Summary 116

6 Simulation Experiments 117
6.1 The Xsim Discrete-Event Simulator 118
6.2 Methodology 120
6.2.1 Common Simulation Elements 121
6.2.2 The Fragmenting Approach 125
6.2.3 The Non-Fragmenting Approach 128
6.2.4 Network Topologies 128
6.2.5 Simulation Parameters 129
6.3 Results 131
6.3.1 Unidirectional Ring 132
6.3.2 Bidirectional Ring 137
6.3.3 Linear Array 137
6.3.4 CDNnet Subset 137
6.3.5 Sensitivity to Workload 145
6.4 Summary 145

7 Conclusions **149**

7.1 Summary and Evaluation 149

7.2 Future Work and Research Directions 150

8 References **154**

9 Glossary **167**

List of Tables

3.1 Message Traffic Summary 64

6.1 Distribution of Message Sizes 123
6.2 Major Simulation Parameters 129

List of Figures

2.1 The Message Handling Environment 12
2.2 An Example Network Organization 15
2.3 A Six Host Network 23

3.1 Network Throughput vs. Offered Load 46
3.2 Unfairness Through Maximized Global Throughput 59
3.3 FIFO Message Multiplexing 61

4.1 Two Interconnected Clusters 73
4.2 A Two-Level Structured Buffer Pool 74
4.3 A Simple S/F Intracluster Buffer System 80
4.4 An Enhanced S/F Intracluster Buffer System 82

6.1 A Single Server Model 120
6.2 CDNnet Subset Topology 130
6.3 Unidirectional Ring: Throughput *vs.* RBSAD 133
6.4 Bidirectional Ring: Throughput *vs.* RBSAD 138
6.5 Linear Array: Throughput *vs.* RBSAD 139
6.6 CDNnet Subset: Throughput *vs.* RBSAD 141
6.7 CDNnet Subset: Throughput *vs.* Transmission Time 142
6.8 CDNnet Subset: Delay *vs.* Message Size 144
6.9 CDNnet Subset: Throughput *vs.* Fragment Size 146
6.10 Bidirectional Ring: Throughput *vs.* Interarrival Time 147

Acknowledgement

My sincere thanks go to Dr. Sam Chanson for his supervision and financial assistance. I am grateful for his patience and gentle but persistent pressure to apply myself to my thesis work when other projects seemed much more interesting. The writing style and organization of this thesis have benefited greatly from his many comments and suggestions over many readings.

I would like to thank my committee members, Dr. Gerry Neufeld, Dr. Son Vuong, and Dr. Erik Skovgaard, for their time and valuable feedback. Many of the basic problems addressed by this thesis arose out of discussions with Gerry. Thanks also to Dr. Paul Gilmore and Dr. Harvey Abramson for serving on my committee early on. I would like to thank Gregor Bochmann of the Université de Montréal for serving as the external examiner and providing constructive criticism.

I would also like to gratefully acknowledge receipt of the MacMillan Graduate Scholarship and the financial support of the Natural Sciences and Engineering Research Council of Canada.

The comradery and friendship of my fellow graduate students in the department have made my tenure as a grad student a pleasant experience. Many people have helped me, some in little ways, some in big ways, for which I am appreciative. Unfortunately, I can only mention a few here.

Extra special thanks go to a few people who made a big difference: Rick Morrison for his companionship, good humour, honest feedback, and support, and to Steve and Shelly Wismath for their friendship, wit (you're so damn cheerful!), and many shared meals.

Thanks to my many office mates over the years for putting up with the occasional bouts of mania and depression that tracked the highs and lows of my thesis work and for just being fun to be around: Marc Majka, Jamie Andrews, Don Acton, Ian Cavers, Murray Goldberg, and Rick Morrison. Thanks guys.

I am grateful to my parents for always encouraging and supporting my academic activities and giving me the freedom to pursue my own goals and interests. I would like to dedicate this work to them.

Chapter 1

Introduction

Not long after computer systems became widely available it was discovered that it would be useful for them to communicate with each other. The study of computer networking followed. Over time, protocols were developed and standards adopted to make communication easier. Concepts such as peer process abstraction and hierarchical network architectures have recently been promoted as means of structuring and simplifying models of networks and networking software. Even so, the study of computer networks has not reached maturity. Most research to date has been concerned with lower networking levels; *i.e.*, levels closer to the hardware. In particular, packet-switching networks have been extensively studied. Higher networking layers and alternatives to packet switching have not received as much attention as they deserve.

When the computer systems constituting a network are not fully interconnected, one means of providing communication between any two computer systems is through store-and-forward transfer. Store-and-forward transfer allows data to be sent between systems that cannot communicate directly with each other. A succession of intermediate systems receive a copy of the data and forward it until the data reaches its destination. This model of data transfer has long been used at the network level as a means of transporting packets and

datagrams. Here, the user (the application level) is presented with the illusion that there is a direct connection to the other communicating entity (*i.e.*, end-to-end communication). The intent is usually to provide a means of communication having low delay so that it appears to the user that there is a direct connection between the two end points. It is not always the case, however, that hosts that wish to communicate share a network level that supports end-to-end communication. Higher networking levels must provide the capability.

Store-and-forward transfer is widely used at the application level to send "electronic mail". In general, the electronic mail system comprises hosts that do not share a single network level and therefore end-to-end communication between all pairs of hosts is not supported at the network level. Electronic mail must therefore be relayed from one computer to the next, by application level entities, until it is delivered to each recipient. The term *application level, store-and-forward message transfer* is used for the technique of store-and-forward transfer of a message in an environment where network level end-to-end communication may not always be possible. The *message transfer system* is responsible for transporting electronic mail from one end point to the other.

The basic task of a message transfer system is to distribute a sequence of bytes provided by a user from one location to one or more locations. The message transfer system is not concerned, or at least should not be concerned, with exactly what the user is transferring. It is, therefore, application-independent. It may be called on to transfer a small interpersonal message or to distribute a very large file.

In contrast to the large body of research aimed at the lower networking levels, there has been little work focussed on the fundamental problems of application level store-and-forward transfer. In particular, flow and congestion control aspects of these systems have not been studied. As both the number of users and applications of messaging increases, so

does the need to address performance issues. This thesis presents a framework for designing message transfer systems that addresses some of these issues.

From a higher level vantage, a network can be viewed as an administrative or communication domain rather than simply as a collection of computers that communicate using some common low-level protocol. It is typically this latter, myopic image that comes to mind, however, when the word "network" is used. Interconnection of disparate networks is becoming widespread. Message transfer systems provide a relatively simple means of interconnecting individual administrative domains.

The advent of international standards for message handling systems is a first step in moving from an environment consisting of largely autonomous computer systems providing electronic mail service to a more homogeneous and cooperative environment [Redell83]. The design methodology introduced in this thesis advocates certain forms of organization and techniques that create a coherent communication system out of what are currently loosely coupled collections of computers (sometimes called a *federation*). The resulting systems are well suited to the task of internetwork message transfer.

This thesis emphasizes *practical* solutions to some of the problems posed by message transfer, primarily those related to resource management. For example, parts of this thesis discuss various buffer management schemes designed for packet-switching or datagram networks and evaluate their use in the message handling environment. While some of these schemes might function in the message handling environment, it is argued that they are impractical.

The main alternative to store-and-forward message transfer is connection-based transfer. It is not the position of this thesis that store-and-forward message transfer should replace connection-based systems. They can be seen as complimentary approaches. In

many circumstances, however, connection-based transfer is not possible or is inferior in some respect.

In the last few years much effort has been put forth to create standards for message handling systems. In particular, the CCITT's X.400 series of recommendations[1] define message formats, protocols, and a model for message transfer. This thesis deals with many areas not addressed in the standardization efforts.

1.1 The Thesis

The major goal of this research is to investigate message transfer systems and aid the design of systems that:

- are able to transport arbitrary length messages via fragmentation,
- are largely independent of network connectivity and transmission media so that they can adapt to network reconfiguration and the introduction of new technologies,
- are deadlock-free,
- are fairly efficient in terms of operational costs (resources, communication, labour),
- offer reasonable delivery times,
- perform well under load, and
- are "fair" in that no users receive undue preferential treatment due to their location in the network, traffic patterns, or network control parameters and algorithms.

Four major contributions of this research can be identified:

[1] The ISO variant of the X.400 recommendations is known as ISO 100021-1 (Information Processing Systems – Text Communication – Message Oriented Text Interchange Systems [MOTIS]).

1. A characterization of store-and-forward message transfer, its advantages, disadvantages, and special problems is given. Such a characterization does not appear in the literature and seems long overdue.

2. A new deadlock prevention scheme, especially suited to the message handling environment is presented. Existing schemes, intended for lower networking levels, suffer many drawbacks that make them impractical for the message handling environment.

3. A structure for the design of message transfer systems is developed. Existing systems have been implemented, typically not through a careful analysis of the basic problems, but rather using simple, unorganized, and *ad hoc* techniques.

4. A simulation study was undertaken to compare the performance of message transfer systems comprised of major elements of the proposed framework to that of current systems.

This thesis is not concerned with the details of a particular design but with the more general task of how to design message transfer systems. A particular design must take into consideration a large number of details, many of which are dependent on the environment in which the system must operate.

Much of this thesis deals with efficient resource management, in particular, buffer space and communication link usage. It might be argued that cheaper memory and faster communication devices will eventually eliminate the need to be concerned with resource usage. This will probably not be the case as ever increasing capacities and performance will simply lead to growing use. Increasing numbers of users and applications will impose more and larger demands on computer systems. It seems likely that the classic time-space tradeoff will continue to be brought into play as technology improves. Further, the possibility al-

ways exists for accidental or malicious misuse of resources. Since the supply of resources is finite, some form of management is required. From a theoretical perspective, increased capacities merely postpone problems such as congestion instead of eliminating them.

It might also be argued that universal connection-based communication will some day be possible, eliminating the need to use store-and-forward transfer techniques. While this may come to pass, as long as there are independent networks and only partial connectivity, store-and-forward transfer will be necessary. There are many technical, economic, and political barriers to be removed before full interconnection can occur. Some aspects of this thesis concerned with resource management are applicable even to fully interconnected networks.

1.2 Overview of the Rest of the Thesis

Chapter 2 reviews the standard model of message transfer and characterizes store-and-forward message transfer. Limitations and difficulties presented by this model are discussed. The application level store-and-forward message transfer paradigm is compared to that of connection-based file transfer.

Chapter 3 focusses on the major problems that must be addressed in store-and-forward message transfer. These include message fragmentation, buffer management, deadlock control, flow control, congestion control, and fairness. The relevant literature is surveyed.

A new, hierarchical scheme for deadlock prevention is presented in Chapter 4. This scheme is well suited to the message handling environment but is also applicable to lower networking levels. The suitability of existing deadlock prevention schemes to the message handling environment is analyzed.

A structure for the design of message transfer systems is detailed in Chapter 5. The

framework deals with basic concepts, general purpose approaches, and tradeoffs that must be weighed in any design. The chapter is mainly concerned with problems related to resource management, such as flow control and congestion control. Some aspects of the design of message transfer systems are not included (*e.g.*, message routing and security considerations).

Discrete-event simulations were performed to study performance aspects of several of the most important mechanisms proposed in Chapter 5. The mechanisms were investigated under a range of traffic conditions and in various network configurations. Details of the experiments, results, and conclusions are the subject of Chapter 6.

Chapter 7 is concerned with problems that remain open and interesting research directions prompted by this research. A summary and evaluation of the significance of the thesis is presented.

Although an attempt has been made to make the thesis reasonably self-contained, a familiarity with basic elements of computer networking and operating systems is assumed throughout. In particular, an understanding of the OSI Reference Model and the concept of networking layers is important [Day83]. Although much of the X.400 model's terminology and many of its concepts are used in this thesis, comments are not restricted to the X.411 Message Transfer Layer Service, but rather refer to message transfer systems in general. Abbreviations and acronyms have been collected in a glossary.

Chapter 2

Store-and-Forward Message Transfer

This chapter introduces the message handling environment and describes the basic operation and characteristics of application level [Bartoli83, Day83] store-and-forward (S/F) message transfer systems.[1] For the sake of brevity, application level S/F message transfer systems will be referred to simply as *message transfer systems* or *messaging systems* and the technique as *message transfer* or *messaging.* Terminology and functionality are borrowed from the X.400 model of messaging.[2] The structure of the EAN Message Handling System [Neufeld86], a system which reflects the state of the art, is outlined. Some of the limitations of current message transfer systems are listed. The primary alternative to S/F message transfer is connection-based file transfer. These two approaches are compared and contrasted.

[1] An earlier version of this material appears in [Brachman88a].

[2] The CCITT recommendations on message handling systems consist of several separate proposals, each dealing with a specific aspect of message handling. Unless otherwise specified, they will be referred to collectively by the name of the first recommendation, X.400. Differences between the various versions and drafts of the X.400 recommendations have little influence on this thesis. When it is necessary to make a distinction, the earlier version will be referred to as X.400/1984 and the more recent version as X.400/1988.

2.1 The Message Handling Environment

The transfer of an application level message from an *originator* to a *recipient* by transferring the entire message from one host to the next until the message arrives at the final destination is called S/F *message-based transfer*. A message may be addressed to any number of recipients (*i.e.*, it may be a *multirecipient* message). The entity that is responsible for storing and forwarding messages is the *message transfer agent* (MTA). The *subnet* interconnects the hosts that constitute the network and carries messages from host to host. The subnet and the MTAs collectively make up the *message transfer system* (MTS). The MTS provides the general, application-independent, store-and-forward message transfer service.

Two MTAs are said to have a *message connection* if there is some S/F path between them. An important characteristic of S/F transfer is that there need not be a continuously available connection between any two MTAs. A pair of MTAs that can establish an end-to-end connection is said to be *adjacent*, with one MTA being the *sender* and the other the *receiver*. There may, of course, be hosts that do not act as MTAs but rather participate in network level data transfer between MTAs.

A *reliable transfer server* (RTS) is the part of an MTA that establishes a session with an adjacent MTA and provides an error-free communication channel called an RTS connection. Such features as checkpointing (synchronization) and recovery, bidirectional transfer, quality of service specifications, multiplexed transport connections, and error detection and recovery may be provided by underlying protocol levels. The connection may physically pass through other nodes, hosts, or even networks. Multiple physical connections between adjacent MTAs may be supported. This permits a tradeoff between cost and delay. For example, an MTA may delay forwarding a message over a telephone line until a lower toll

rate takes effect instead of using a more expensive X.25 circuit immediately.

In the simplest case, end-to-end communication in the message handling environment is observed at two levels. The higher level communication is between the originator of a message and a recipient. The lower level communication is between two adjacent MTAs over an RTS connection. The following terms are used to identify the end points of the two levels of end-to-end communication:

Originating MTA	–	the first MTA to receive the message and become responsible for forwarding the message to one or more adjacent MTAs
Recipient MTA	–	an MTA that services one or more of the recipients specified by the message
Sending MTA	–	an MTA that has stored a message and is in the process of forwarding it
Receiving MTA	–	an MTA that receives a message that it will store and subsequently forward, deliver to a local recipient, or both.

Communication between a pair of MTAs involves four phases: session establishment, negotiation, data transfer, and release. One MTA establishes an RTS connection to the other, either because it has one or more messages to send or because it is polling to see if the other MTA has messages to send. Once a connection is established, negotiation may take place to determine which message or messages, if any, to transfer. In the usual case, after a message has been successfully forwarded the receiver becomes responsible for the message and the sender may delete its copy. In some cases negotiation simply consists of taking turns transmitting messages. More sophisticated MTAs might support repeated negotiation and transmission phases. The RTS connection is released when no more messages can be sent. An MTA may allow several simultaneous RTS connections to another MTA. There may also be simultaneous RTS connections to different MTAs.

2.1.1 The X.400 Model

This section provides an overview of the terminology and model of message handling specified in the X.400 recommendations on message handling systems [CCITT88a, Cunningham83]. Both Sadowski [Sadowski84] and Koorland [Koorland85a] provide more complete descriptions of the X.400/1984 recommendations and their application to tasks other than interpersonal mail.

In the X.400/1988 model, a user is either a person or a computer process. The user agent (UA) is an application program that assists the originator in preparing messages to be submitted to the MTS. The message handling system (MHS) may be indirectly used through an access unit (AU) that links another communication system to the MTS. The message store (MS) acts as a storage facility for messages and is an intermediary between a UA and an MTA. The collection of UAs, MSs, AUs, and MTAs makes up the MHS. The MTS relays and delivers messages to the intended recipient UAs, which then make the messages available to the intended recipients. The MHS and all of its users are collectively referred to as the message handling environment (MHE) (Figure 2.1).

The primary purpose of the MTS is to transport information objects called *messages*. A message consists of an arbitrary length *envelope* and an arbitrary amount of *content*.[3] The envelope carries information to be used by the MTS when transferring the message; *e.g.*, the list of recipients to whom the message is to be sent. The message content is used by UAs and AUs[4] and is the information that the originating UA wishes delivered to one or more recipient UAs. Within the X.400 recommendations, the term *protocol* denotes those rules that govern the transmittal of messages, and the syntax and semantics of their

[3] This is not exactly true for X.400/1988. There is a limit of 32767 recipients and $2^{31} - 1$ bytes of content.

[4] An MTA may convert the content from one format to another, however.

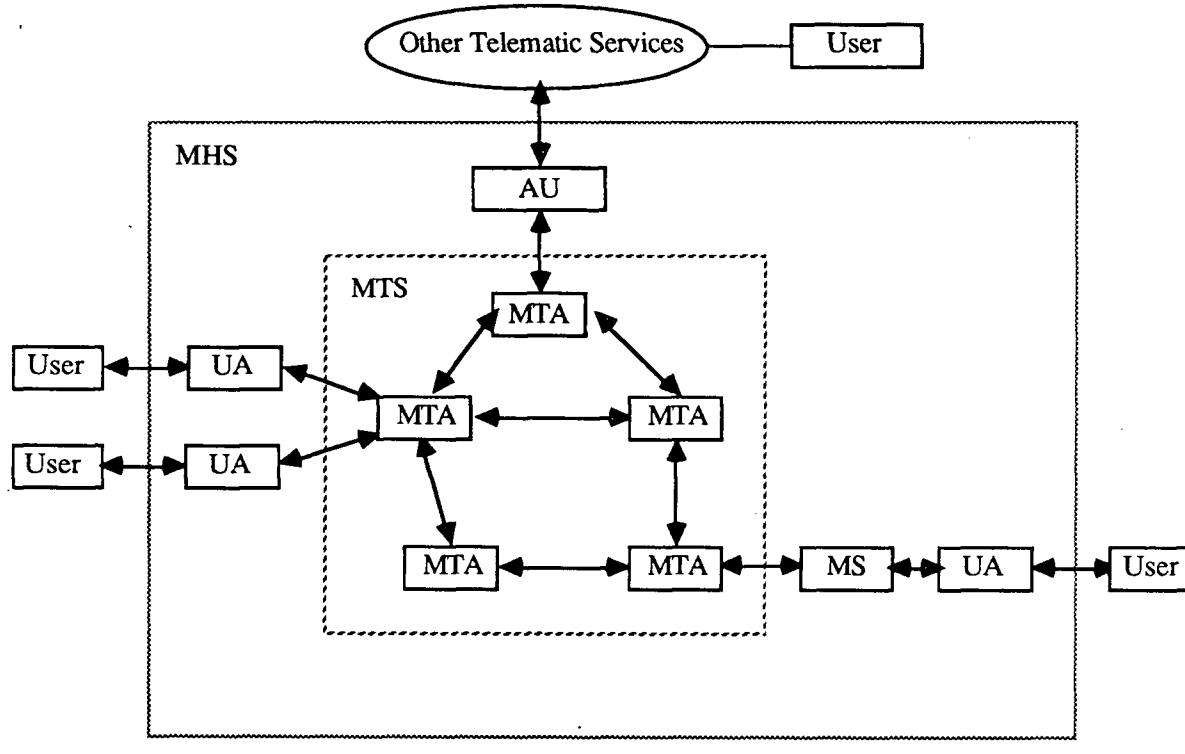

Figure 2.1: The Message Handling Environment

contents. Both the envelope and its contents are formatted according to a given protocol. The envelope is structured according to the protocol specified in Recommendation X.411 [CCITT88b]; the content may be structured according to whichever protocol best suits a particular application. UAs that cooperate with each other by using the same message protocol form a *class* of UAs.

A second type of information object, a *probe*, is sent to a recipient's MTA to determine the deliverability of a *described message*. A probe has no content but its envelope is similar to that of a message. The probe's envelope indicates the length of the content of the described message.

In addition to conveying messages generated by a UA, MTAs may themselves generate a third type of information object, a *report message*, for delivery by the MTS. Report messages are generated upon request of the originating MTA or in response to an exception. Two examples are the non-delivery report message and the confirmation of delivery report message. A non-delivery report message is delivered to the originating MTA of a message that could not be delivered to an intended recipient. When submitting a message, the originating MTA may request that a report message be returned when the message is delivered.

Every user of the MHS is assigned at least one identifier, called an originator/recipient name (O/R name), that unambiguously identifies the user in the MHS. An originator/recipient address (O/R address) is an attribute list that helps the MTS locate the user's point of access to the MHS.

Message handling functions can be considered to be divided into two layers: the user agent layer (UAL) and the message transfer layer (MTL). The two corresponding functional entities are the user agent entity (UAE) and the message transfer agent entity (MTAE).

The UAE deals with the representation of the message's contents and other cooperating UA layer functionality. An originator or recipient above the UAL is able to use the capabilities provided by the cooperation of the UAEs and MTL below. The MTAE supports the layer services of the MTL in cooperation with other MTAEs.

2.2 Characteristics of Message Transfer Systems

Messaging systems have several characteristics that differentiate them from other forms of data transport. Major characteristics of message transfer systems include the following:

1. It is not necessary to establish an end-to-end connection (originator to recipient) to transfer a message. Only a message connection is required.

2. There are few restrictions on the network topology[5] and the network may be heterogeneous. Many different types of transmission media (*e.g.*, telephone, Ethernet, common carrier, leased line) may make up the subnet. It may be the case that only one MTA of a receiver/sender pair is capable of initiating a connection.

3. A possibly indeterminate (but finite) delay may be incurred before a message can be forwarded. This delay may range from less than a second to several days.

4. Adjacent MTAs may use any reliable protocol (*e.g.*, ISO Transport, TCP/IP) to transfer messages.

5. In X.400/1988, recipients of a message may be indirectly specified in a "distribution list" or "mailing list" containing O/R names. A message sent to the distribution list

[5] The term *network topology* refers to the way the hosts forming a network are interconnected, including adjacency information, types and availability of transport connections, and transfer rates. A *heterogeneous network* consists of hosts of varying type and capacity, not necessarily running the same operating system.

is forwarded to each O/R name in the list. Distribution lists can be nested; *i.e.*, they can contain the O/R names of other distribution lists.

Figure 2.2 depicts a simple network consisting of nine MTAs interconnected in various ways. A message sent from MTA_0 (the originating MTA) to MTA_4 (the recipient MTA) might be forwarded through MTA_1, MTA_2, and MTA_3 before arriving at MTA_4.

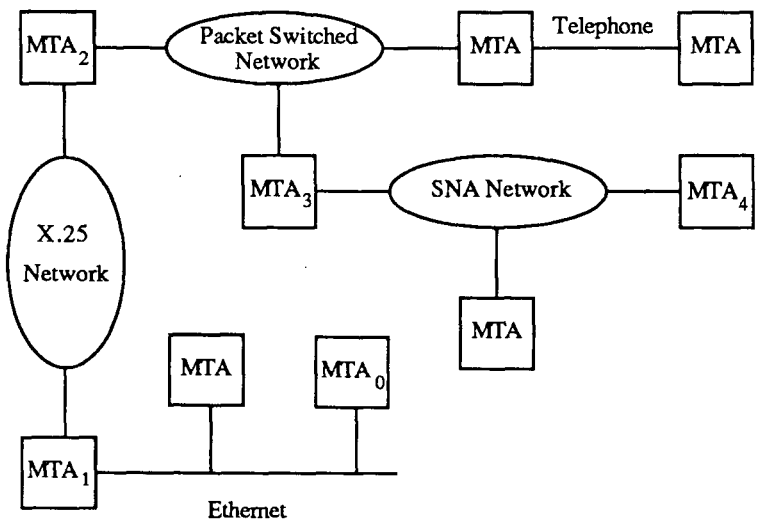

Figure 2.2: An Example Network Organization

Electronic mail systems are the most common application of message transfer but the technique is also used by file transfer applications [Sadowski84, Brachman86] and to access remote databases [Koorland85a, Koorland85b]. Other applications that can take advantage of the offline nature of message transfer, including interlibrary loan systems, are under development.

An important difference between application level and network level transfer is that the former must store messages on secondary storage to be able to support reliable transfers in the presence of failures. Therefore, references to buffer space in the context of message transfer implies the use of disk storage space.

Application level messages, as implemented in current messaging systems, can be viewed as arbitrary length, multirecipient datagrams. Like the datagram environment, there is a single submission event and a single delivery event. Once accepted by the message transfer system, a message will be delivered unless either of two events occur: the message expires or is destroyed by some action outside the control of the system (such as the failure of an MTA). Not all of the intended recipients of a multirecipient message may receive the message.

The more likely event is that the message could not be delivered within the *message lifetime*. The message lifetime is the sum of the maximum length of time a message is given to traverse the network and the maximum length of time a recipient is given to accept the message after it arrives (the *acceptance period*). In many current systems the message lifetime is often simply chosen to be a "big enough" value. The acceptance period may vary among different MTAs. Sometime after a message arrives at its destination it is "secured" and is considered to be outside the message handling system and not subject to destruction by the system. Typically, this involves an application accepting a copy of the message and subsequent removal of the message from a spooling area (*i.e.*, some secondary storage partition reserved for message storage). If a message is discarded because it is too old, the originator may optionally be notified that the message could not be delivered.

The other event is a *catastrophe* that destroys the only copy of the message in the message handling system. It is assumed that through the use of synchronization methods

and disk storage with routine backups only a rare event can cause the destruction of a message within the message handling system. Magneto-optical memory, WORM (write-once, read-many) memory, or stable storage can be used to provide higher reliability than traditional magnetic disk memory. In certain circumstances a message may be discarded if a copy exists.

Of primary interest are network topologies that have a hierarchical organization. This is typical of wide-area networks. The hosts constituting a university campus, an organization, or a city may be grouped together to form a *cluster*.[6] Each host normally belongs to a single cluster. Communication within a cluster of hosts is typically fast (*e.g.*, over a local area network) or cheap (*e.g.*, using telephone lines), while communication between clusters is typically slower and more expensive (*e.g.*, public carrier, leased line, long distance telephone). The number of clusters in a network will tend to be much smaller than the total number of hosts. A common example of such topologies are message handling systems where, typically, a small number of *cluster gateway* MTAs within each cluster serve as connections between the cluster and the rest of the subnet. An MTA that does not act as a cluster gateway is called an *internal* MTA. Few assumptions can be made regarding the topology of the subnet: message handling systems are often implemented on top of existing hardware rather than being designed from scratch.

2.3 Current Message Transfer Systems

Many electronic mail systems use the S/F technique to transfer electronic mail. Examples of widely-used systems are UUCP-based systems [Nowitz78, Nowitz86], MMDF [Crocker79, Kingston86], Sendmail [Allman86], SMTP-based systems [Postel82], and EAN

[6] A similar notion of clusters is used by the Diamond message system [Thomas85].

[Neufeld86].

The EAN messaging system is representative of a state of the art message handling system. An overview is presented in the next section.

2.3.1 The EAN Message Handling System

EAN [Neufeld86] is a store-and-forward message handling system developed at the University of British Columbia based on the X.400/1984 recommendations. EAN forms the basis for CDNnet, a Canadian academic network, as well as many European networks [Quarterman86]. It has been ported to a number of different machine architectures and operating systems. EAN includes both MTA and UA functionality. Gateways exist to EAN-based networks and other message handling systems.

In the current implementation, an EAN MTA does not determine the entire route a message should take. Instead, MTAs use static routing tables to determine only the next adjacent MTA to which the message should be forwarded. EAN forwards the entire message at each hop along the route and there is no predetermined maximum message size. Messages stored at an intermediate MTA share a common buffer area on a first-come, first-served basis. Under certain circumstances a message can expire and be discarded by an intermediate MTA.[7] Some of the consequences of these design decisions are discussed in subsequent chapters.

EAN can use several types of network connections and supports several protocols for network level connections, including X.25, TTXP,[8] TCP/IP, and DECnet. The system

[7] A surprising side effect of this has been observed. If a message from a non-X.400 MHS passes through an X.400 gateway it is possible for it to be expired immediately. In some major message handling protocols, the envelope does not contain the creation date of the message, and although user-specified date information in the header is not always trustworthy, EAN uses it anyway [Demco88].

[8] TTXP is a half-duplex protocol based on the MMDF [Crocker79] protocol. It is used by MTAs that have terminal access (PAD) to an X.25 network, a connection to the telephone network, or a direct

has no global flow control, congestion control, or deadlock prevention components.

When a message arrives at its recipient MTA it is stored in a shared spooling area. The MTA searches for the recipient's name in a table to determine which program should be notified of the arrival of the new message. This scheme makes it easy to interface the message transfer system with arbitrary applications. An application is expected to eventually accept the message so that it can be deleted from the spooling area. In principle, a message can sit in the spooling area forever.

2.3.2 Limitations of Current Message Transfer Systems

Current message transfer facilities have a fundamental limitation: if any intermediate MTA cannot store a message in its entirety due to insufficient buffer space the transfer fails. Session checkpointing, while a useful mechanism for efficiently continuing a transfer after a failure, does not address the problem of insufficient space to store the entire message. Even if there is enough capacity at an intermediate MTA to store a large message, it could take an indeterminate period of time before enough of this space could be allocated to the message. During the period of time the MTA stores the large message it could be left with insufficient space to handle any other message traffic.

A user's only alternatives are to manually fragment the large message into small pieces or to bypass the message handling system entirely (*e.g.*, by mailing a magnetic tape). Besides being clumsy, the former may result in inefficiencies since a user is not in a good position to choose an appropriate fragment size. It also puts the onus on the recipient to reassemble the message. Worse, it can result in deadlock. The latter alternative is also unattractive because of the inconvenience, media compatibility problems, and so on.[9] A

asynchronous connection to another MTA.

[9] Using the postal service to mail magnetic tapes does make sense in some situations. The aphorism "Never

new mechanism, message fragmentation, is needed to automatically and efficiently transfer messages too large to be completely stored on an intermediate MTA.

No widely used message transfer system offers *reliable* message delivery. A reliable messaging system either delivers a message or notifies the originator that the transfer has failed. As in datagram transport, current messaging systems do not guarantee *message ordering* (*i.e.*, messages sent by an originator may be delivered to a recipient in a different order). This can have important implications for users of the message transfer system.

None of the popular message transfer systems deal with end-to-end issues such as buffer management, deadlock, flow control, congestion control, and fairness. These issues may be addressed on a local basis (*i.e.*, at a particular host or between adjacent hosts) but not on an originator to recipient basis. The message internetwork is, in fact, a mixture of many different types of message transfer systems that cooperate only on a pair-wise basis to forward messages.

2.4 Connection-Based File Transfer Systems

In an environment where connections can be established between all hosts, connection-based file transfer systems offer an alternative to message transfer. Connection-based file transfer has traditionally been performed using a remote login ("virtual terminal") approach. The user (or a process acting on behalf of the user) signs on to a remote system and interacts with it as if the terminal were directly attached to the remote system instead of being attached to a local computer that is able to establish a connection to the remote computer. A file is transferred by issuing a command to the local side of the program which in turn communicates with a remote program. The ISO FTAM standard [ISO86] defines a

underestimate the bandwidth of a station wagon filled with magtapes" comes to mind.

connection-oriented system providing file transfer, access, and management services. The ARPANET FTP [Clopper80], BBN's AUTODIN II [Forsdick80], the Berkeley Unix TIP [Joy83], and Kermit [Da Cruz84a, Da Cruz84b] are examples of virtual terminal based (connection-oriented) file transfer systems.

Teng *et al.* [Teng83] and Aggarwal *et al.* [Aggarwal85, Aggarwal86] have designed a connection-oriented file transfer protocol, adopted as an AT&T standard, that allows offline transfer.

2.4.1 Message Transfer *vs.* Connection-Based Transfer

Message transfer has a number of advantages over typical virtual-terminal based and connection-based file transfer:

1. MTAs incapable of establishing an end-to-end connection can send and receive messages if they have a message connection.

2. There is increased network independence than when end-to-end connections are necessary. Although a message connection may span MTAs that communicate via heterogeneous underlying subnets, no special action needs to be taken as the message is forwarded. If a message is forwarded over an X.25 virtual circuit and then a TCP/IP connection, for example, the message does not need to be converted to a new format. In contrast to this, there are many problems associated with datagram level and virtual circuit internetworking [Sunshine77, Shoch79].

3. Because of its offline nature, it is easier to handle synchronization and error recovery. Connection-based file transfer methods may need to establish, manage, and maintain several connections simultaneously for the duration of a transfer. This may be more

difficult to accomplish when there are many intermediate packet-switching nodes involved since the probability of a failure increases with the number of nodes involved. The next two advantages are direct consequences of this.

4. Third party transfers are simplified. An example of a third party transfer is transferring a file from Host B to Host C by sending a *request* to B from Host A. EANft [Brachman86], a prototype file transfer system built on top of the Unix version of EAN, performs a third party transfer simply by generating a message based on the user's request and sending the request message from the user's host to the EANft server residing on the host possessing the files. This second host then packages the files and sends the resulting message to the EANft server at a specified recipient host.

5. Distribution of files to multiple recipients, through a multirecipient message or a distribution list, is much simpler. From the user's point of view, it is as easy to send a message to multiple recipients as it is to a single recipient. Transferring files to more than one recipient typically requires multiple sessions with a connection-based file transfer system.

6. It is not necessary for the user to login to the destination computer, as is required in many virtual terminal based approaches. This eliminates the need for system administrators to maintain individual accounts for users only wishing to transfer files, and users do not have to learn various login procedures.[10] Since the file transfer system limits the operations a user can perform, it is easy to restrict a user's access to system resources. The same access control mechanism can be used by all hosts on the network to limit access to files [Brachman87].

[10] One popular solution to this problem is to create an "anonymous login" that anyone may use. It has been noted, however, that this idea is inherently dangerous [Joy83].

7. Since the entire connection is offline, "think time" overhead is eliminated and the user does not tie up a terminal waiting for the transfer to complete.[11]

8. Since there is no real-time constraint on delivery of a message, connections between adjacent MTAs may be conveniently made during favourable rate periods or when the MTA is able to perform the transfer more efficiently.

9. It is possible to provide the capability of involving files that are not instantaneously available; *e.g.*, files that are to be read from tape.

Message transfer systems offer certain efficiency advantages over connection-based transfer systems when there are multiple recipients. Figure 2.3 depicts a simple interconnection of six hosts.

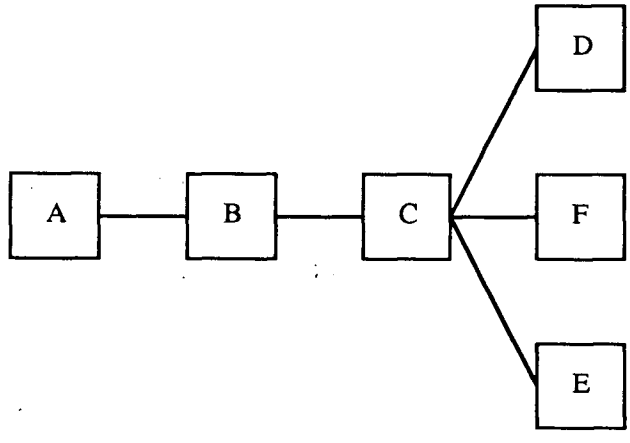

Figure 2.3: A Six Host Network

[11] The transfer need not be offline but given the flexibility of the underlying subnet this is normally the case.

Suppose a user on Host A wants to transfer an N byte message from A to each of D, E, and F. In a message handling environment where only adjacent pairs can establish connections, message transfer will perform five transfers of N bytes each ($A \rightarrow B, B \rightarrow C, C \rightarrow D, C \rightarrow E,$ and$C \rightarrow F$) when a multirecipient message is sent. Each MTA on the route only needs to know the next MTAs to which the message must be forwarded. Five connections, which need not be concurrent, must be established. In a connection-based environment where any pair of hosts can establish a connection, the simplest solution is to establish individual connections to each of D, E, and F.

There are several implementation techniques for storing and forwarding a multirecipient message at a "splitting point" (*i.e.*, an MTA, such as the one at Host C in this example, that sends copies of a message to different MTAs adjacent to it). Multiple versions of the message can be created with the same content but with each envelope specifying a different recipient list. Duplication of the message content can be avoided by creating new envelopes that point to the same content. Creation of multiple envelopes can be deferred until the time a message is forwarded, eliminating additional secondary storage requirements.

The connection-based approach could be improved in terms of the total number of bytes transferred through each host, thereby reducing the volume of network traffic, by transferring the message once from A to C and then from C to each of D, E, and F. A minimum of four connections would be needed and N bytes would flow through each of B and C as in the message transfer case. The originator must generally know or have access to the complete network topology, however, to find the optimal strategy in terms of minimizing the number of connections required and the total number of bytes transferred. With increasingly complex network topology, the task of efficient connection-based distribution of a message addressed to a large number of recipients would become unmanageable.

Furthermore, if originator to recipient connections are not available, all of the problems associated with current message transfer systems would be present. For example, if C is used as an intermediate site it must be able to store the entire message. Without a significantly more elaborate routing system and equivalent buffer management, the connection-based approach does no better than message transfer.

Message transfer has several disadvantages over connection-based transfer. Some of these demonstrate the need for online access in some situations. Conversational access is not precluded in the MHE but is dependent on network topology. A query/response model can be used in the message handling environment to approximate connection-based interactions (*e.g.*, file management tasks). Disadvantages of message transfer include the following.

1. In current systems, the maximum size of a transfer is limited by the smallest amount of available space on any machine involved in the transfer.

2. The originator may not know when the transfer will actually occur after the request is submitted, although it may be possible to associate a priority with a request. A message transfer system can, of course, notify the originator when the transfer completes and the originator may be able to inquire about the status of a particular transfer.

3. Because of its offline nature, there may be some delay between when an exception occurs and when the originator is notified.

4. Several different transactions may be required for a user to perform a particular task that could be done much more quickly by a connection-based system; *e.g.*, obtain a directory listing and then transfer a file appearing in the listing. The query/response

approach may be more wasteful of resources since the result of a management request may return information that the user only wants to read once and not have stored. In a connection-based system, a user is usually able to interrupt a remote application.

5. It is more difficult to abort a request once it has been submitted. Again, a connection-based system usually supports interrupts.

2.5 Summary

An overview of message handling and the message handling environment has been presented. Allowing arbitrary network topology implies that end-to-end network level connections may not be possible and S/F message transfer is required. Therefore, techniques making heavy use of end-to-end acknowledgements are impractical.

Unique aspects of messaging include:

- End-to-end network level connections between originator and recipient are unnecessary.
- Multirecipient messages are supported.
- Variable length and very large messages may be transferred.
- Multiple physical connections between adjacent MTAs are possible. This allows a tradeoff between cost and delay.
- Distribution lists are easily supported.
- Secondary storage is used to buffer messages.
- The message handling system may span many heterogeneous networks.

Although message transfer offers a number of advantages over connection-based transfer, current systems have a fundamental limitation and may suffer from performance and fairness related problems. These problems are more closely examined in the next chapter.

Chapter 3

Issues in the Design of Message Transfer Systems

The design of message transfer systems that offer improved performance and fairness characteristics over current systems necessitates a study of the major components of these systems. It is argued that message fragmentation, buffer management, deadlock management, flow control, and congestion control are required to increase the reliability, flexibility, robustness, and performance characteristics of message transfer systems. This chapter looks at these components, surveying related research at lower networking levels and identifying problems that currently have no satisfactory solution. Subsequent chapters expand on these ideas and introduce solutions. Other issues, such as reliability, routing, and security, are beyond the scope of this thesis and are not addressed.

After a discussion of application level message fragmentation in Section 3.1, the management of store-and-forward buffers is addressed in Section 3.2. Deadlock, flow control, and congestion control are examined in the three subsequent sections. Section 3.6 looks at the fair management of network resources and the tradeoffs involved. A study of message traffic is presented in Section 3.7 and the chapter concludes with a summary. Before discussing the design issues, two important concepts are introduced.

There is a need in a distributed system, such as a message handling system, to distribute routing or tuning information, perform control functions (*e.g.,* acquire or release buffer space), or to query remote systems for performance, statistical, or state information. To a great extent, a message handling system can use itself to accomplish this. A message sent by one MTA to another to perform some function internal to the message handling system is called a *control message.*

It is convenient for a message handling system to consider messages that are related to each other in some way to belong to a class of messages. Messages sharing some characteristic, such as originating user, originating cluster, or priority level are said to belong to the same *traffic group.* A message can belong to more than one traffic group.

3.1 Message Fragmentation

Fragmenting a message involves breaking it into two or more pieces, each of which is also a message (*i.e.,* a fragment consists of an envelope and some portion of the original message's content).[1] No widely used message handling system currently performs message fragmentation.

When buffer space is at a premium, fragmentation allows it to be shared more equitably since large messages need not be stored in their entirety at intermediate MTAs.

Message fragmentation can provide other benefits. First, fragmentation allows for reduced transfer time by increasing parallelism in the transfer of a message through "pipelining": while an MTA is receiving fragment number N it may be able to forward fragment $N-1$ to the next MTA. Pipelining may also result in much lower buffer occupancy (storage-time product) at intermediate MTAs. For example, suppose a large message arrives at an

[1] An alternate approach is discussed in Section 5.2.1.

intermediate MTA but the only link to the next MTA on the route goes down before the message can be forwarded. The large message takes up buffer space that could be used for other purposes at the intermediate MTA. In such a case, fragmentation, in conjunction with an appropriate buffer sharing policy, might result in at least some of the fragments being routed around the intermediate MTA. Second, it may be possible for fragments of a message to follow different routes in parallel, thereby reducing the transfer time for the original message. Also, MTAs that cannot establish multiple physical or virtual connections to adjacent MTAs may use fragmentation to reduce mean queuing delay by effectively time-sharing communication channels. For these reasons it may be desirable to fragment messages even if sufficient space is always available. These ideas are also relevant to packet-switching networks.

Fragmenting messages introduces additional complexity to the MTA. As well as keeping track of individual fragments, an MTA must deal with confirmation of delivery and exception reports; *e.g.,* the originator should receive a single confirmation of delivery report when the last fragment is delivered to a particular recipient. The handling of the expiry date of a message must be modified to prevent the first arriving fragments from expiring before the remaining fragments arrive. Also, an MTA must be able to detect the loss of a fragment.

Some results obtained for transferring packets and datagrams through a subnet, such as algorithms developed to prevent S/F buffer deadlock, perform flow control, and manage buffer allocation, may be adapted for use at the application layer. Packet fragmentation, both intranet and internet, has been extensively studied [Sunshine77, Shoch79, Bennett82]. Although fragmentation at the application level has some aspects in common with network and data link level fragmentation, there are also a number of important differences:

1. Message connections are used rather than network connections.

2. There is a wide variance in message size. A message may be of arbitrary length whereas the maximum packet or datagram size is usually fixed.

3. At lower protocol levels the network can safely and economically discard a packet knowing it will be retransmitted. At higher levels, however, the unit of information hopping from MTA to MTA is a message that is typically 10 to 1000 times larger than the lower level packet (see Table 3.1, page 64) and is therefore too expensive to discard. Furthermore, once an MTA accepts a message it becomes responsible for delivery of the message. This implies that the message must be saved on secondary storage until it can be forwarded or delivered. Current message transfer systems are reluctant to discard a message since there is often no mechanism to retransmit the message.

3.2 Buffer Management

Buffer management, flow control, congestion control, S/F deadlock control, and message fragmentation are closely related issues [Gerla80]. For example, a buffer management policy can be designed so as to provide both flow control and deadlock prevention. Also, the choice of fragment sizes may depend on the buffer management policy.

When choosing or designing a buffer management policy several aspects must be considered:

- fragment size
- flow control and congestion control
- S/F buffer deadlock

- reassembly and gateways
- layering concerns

3.2.1 Fragment Sizes

Messages may be fragmented into fixed size pieces, variable size pieces, or a hybrid scheme may be used. The fragment size may be determined *a priori* for the entire network or negotiated by adjacent MTAs at session establishment time. The implications are discussed below.

3.2.1.1 Fixed Size Fragments

Fixed size fragments offer the advantages of easier buffer management and possibly avoiding inefficiencies caused by repeated fragmentation and reassembly.[2]

Two alternatives are:

1. Globally fixed - the network management decides on a maximum fragment size, presumably the largest size that the most limited MTA can deal with. All messages, except possibly the last, will be of the maximum size. This type of fragmentation is referred to as *source fragmentation* since only the originating MTA is responsible for performing fragmentation.

   A fixed size may be inefficient when a transfer involves MTAs with available capacity much larger than the fixed fragment size. Also, if there is insufficient room on an intermediate MTA to store an entire fragment, then none of it is transferred.[3]

[2] This is also applicable to datagram fragmentation.

[3] If both MTAs support session checkpointing some of the fragment might be transferred and the session reestablished later. The partial fragment could not be forwarded, however.

2. Globally fixed range - the network management decides on several fragment sizes (say, N bytes, $2N$ bytes, and $4N$ bytes). A receiving MTA informs the transmitter of the size it is willing to accept. If the current available buffer space is less than the smallest size it doesn't accept any of the message. A fragment of size $4N$ may subsequently be fragmented into 2 fragments of size $2N$ or 4 of size N.

3.2.1.2 Variable Size Fragments

1. Entire message - no fragmentation. This is, of course, the simplest strategy but is unsatisfactory for the reasons given earlier.

2. As large as possible - the receiving MTA accepts as much of the message as it can. The advantage of this approach is that the mean number of messages that need to be transferred between adjacent MTAs is minimized. This requires negotiation of the fragment size between the sending and receiving MTAs.

3.2.1.3 Hybrid

These strategies require negotiation of the fragment size.

1. Locally fixed - a maximum message size is assigned to individual MTAs. When a connection is established, the receiving MTA informs the sender of the maximum size it will accept. The result is that adjacent large capacity MTAs can exchange large messages but that these large messages may subsequently be split into many fragments if a smaller capacity MTA is on the route.

2. Locally variable - the size is determined at each connection establishment and may change from connection to connection. When a sending MTA establishes a connec-

tion to the receiving MTA in order to transfer a message, the receiving MTA may determine the size it will accept (from nothing to the entire message) based on its current state. The receiving MTA may be provided with information on the size and priority of the message as well as the route the message has taken so far (including the identity of the originator). In determining the message size it will receive, the receiving MTA might take into account any of:

- how much buffer space is already being used by messages sent by the sending MTA
- how much buffer space is already being used by messages sent by the originating MTA
- how much buffer space is already awaiting transmission to the same adjacent MTA as the new message (if the receiving MTA can easily determine the route the new message will take)
- distinctions between "levels of service". The level of service indicated by the originator of the message (*e.g.*, "low", "normal", "high") and the message type (*e.g.*, report message or control message) may be taken into account.

The message size may be dependent on the transmission rate. With lower transmission rates smaller message sizes may be favoured to lower delay for other messages.

3.2.2 Buffering Strategies

There are a number of ways an intermediate host could manage the *transit buffers* used to store messages that must be forwarded. These buffer sharing schemes may be static or adaptive. Adaptive schemes take into account the changing state of the network whereas static schemes do not.

The following two sections discuss buffer management schemes for packet-switched networks. Some of the goals and requirements of buffer management in the message handling environment are outlined in Section 3.2.2.3.

3.2.2.1 Static Buffer Management

Buffer management in Cigale, the datagram packet-switching subnet of the Cyclades network [Pouzin73], is based on two mechanisms. An output queue length limit imposes the same maximum queue length on all output lines, discarding packets arriving at a full queue. Packets that arrive when there is no buffer space are acknowledged and then discarded, leaving retransmission to the end-to-end protocol.

Irland [Irland78] presents a model for allocating buffer space from a buffer pool and derives the computationally simple *square-root sharing policy.* This policy sets the maximum number of buffers for each communication link to the total number of buffers divided by the square root of the number of links.

Kamoun and Kleinrock [Kamoun80a] model five methods, called channel queue limit (CQL) flow control schemes [Gerla80], for sharing a pool of buffers among a set of communication channels and discuss their relative merits:

1. *complete sharing* (CS) shares buffers among all adjacent MTAs on a first-come, first-served basis

2. *complete partitioning* (CP) permanently divides the available buffers among the channels

3. *sharing with maximum queue lengths* (SMXQ) imposes a limit on the number of buffers that may be allocated at any time to any channel

4. *sharing with a minimum allocation* (SMA) allocates a minimum number of buffers to each channel and allows the remaining buffers to be shared without further constraint

5. *sharing with a maximum queue and minimum allocation* (SMQMA) is SMA with the additional constraint of a maximum buffer allocation

Kamoun [Kamoun81] proposes a mechanism called *drop and throttle flow control* (DTFC)[4] that gives higher priority to transit traffic (*i.e.*, traffic in transit) over new traffic. If the number of buffers in use at a particular host exceeds a limit value, then new traffic is rejected in favour of transit traffic. When all buffers are in use, transit traffic may be discarded.

Given the blocking requirement for each channel, Yum and Dou [Yum84] determine the minimum buffer size and the best strategy among CS, CP, SMXQ, and SMA that achieves this minimum under various conditions.

3.2.2.2 Adaptive Buffer Management

For his model of a packet switch, Irland [Irland78] derives the *optimal restricted buffer sharing policy*. The optimal choice for the maximum queue length for each of N links (optimal in that it minimizes the overall probability of losing any packet) is obtained from closed-form algebraic formulae given the load of each link.

The problem of updating buffer allocations in a CP policy as the traffic load changes is addressed by Thareja *et al.* [Thareja82]. A heuristic is described that determines when a change in partitioning is needed based on the current traffic load of each partition and changes in system throughput since the last update. Tipper and Sundareshan [Tipper88]

[4] The same mechanism is called *sharing with discrimination and maximum queue length* (SDMXQ) in [Kamoun80b].

formulate the optimal buffer allocation problem in a mathematical optimization framework and outline several procedures for solving the problem. A simple analytical approximation to the optimal buffer management policy, valid under moderate to heavy loads and offering superior performance, is given.

Thareja *et al.* [Thareja83a, Thareja83b, Thareja84a, Thareja84b] describe a class of buffer allocation policies called *delayed resolution policies* (DR) and compare the buffer occupancy delay incurred by DR policies with that of non-DR policies. In DR policies, messages are accepted whenever the buffers are not full. When a message arrives and the buffers are full, a decision is made whether to accept the arriving message and discard another. The decision is based on the state of the system and upon its past history. DR policies can adapt to a changing load environment by changing the number of buffers assigned to each link.

3.2.2.3 Buffer Management in the Message Handling Environment

The following goals are set out for a buffer management policy in the message handling environment:

1. The policy should be independent of the routing algorithm being used, but should be able to offer improved performance if information, such as network connectivity, is available. Changes in the routing algorithm should not require significant changes to the buffer management policy.

2. The advantages of message fragmentation should be preserved.

3. It should operate well under various traffic conditions – continuous or bursty arrivals and submissions, a wide range of data transfer rates, and high percentage of short

messages and low percentage of very long messages. Network traffic patterns are discussed in Section 3.7.

4. It should have relatively low overhead.

5. It should be reasonably simple to understand, model, and evaluate.

While it is possible to use a static buffer management policy in the message handling environment, because of the nature of the traffic, it is expected to behave poorly. Since it is too expensive to discard a message once it has been accepted,[5] DR policies are unacceptable. Using some other type of adaptive policy is attractive, however.

Since in the message handling environment the unit of data transferred between MTAs is the message, it is possible for an adaptive policy to use information concerning the state of a particular transfer in addition to the state of adjacent MTAs. Because messages are submitted like datagrams, in a single submission event, the size of a message is known at the time of submission and, as far as the message handling system is concerned, the message is not related to any previous or future submission unless message ordering is enforced. This is in contrast to stream-based transport level fragmentation [Chapin82, Chapin83]. The path a message has taken is available since each MTA inserts its name and a time stamp into the envelope when it obtains the message.

A scheme for adaptively sharing transit buffers might be based on any of the following techniques:

- Adaptive partitions

  An obvious scheme is to extend SMQMA so that the maximum and minimum alloca-

[5] Although an MTA may discard a fragment if it already holds a copy of the fragment.

tions are dynamically adjusted. Current and estimated near-future conditions would be considered when periodically readjusting allocations.

- Credit-based

  In a credit-based scheme, the sender obtains buffer space credit from the receiver. When the credit is consumed, the connection must be reestablished at a later time to transfer more of the message. The receiver may provide a time and date for the next session. Credits must have a limited lifetime so that space that turns out to be unused is not reserved indefinitely. A benefit of credits is that polling between sending and receiving MTAs can be reduced.

- Message stream

  The message handling system can treat fragments of the original message as being related. One can view the sequence of fragments going from the originating MTA to the recipient MTA as a stream. If all fragments follow the same path, then MTAs along the route can use this information to prepare in advance for the buffer space requirements of the transfer. If this scheme is viewed as a type of credit-based scheme where the reservation of space is implicit, then the lifetime of the stream, as seen by an intermediate MTA, can be limited so that resources are not reserved indefinitely.

  As in some virtual circuit flow control schemes, an upper limit could be imposed on the amount of space used at an MTA by a particular message stream. This can be used to throttle submissions of new messages to the network due to "backpressure" propagating through intermediate MTAs. This flow control scheme is deadlock-free if buffers are dedicated to a message stream when it is set up. It eliminates the ability to route messages independently, however.

A system might allow refragmenting; *i.e.*, fragments may themselves be fragmented.

The buffer management strategy may take into account, for each adjacent MTA, such characteristics as the number of simultaneous connections that can be established, the transfer rate on each connection, the period between transfer sessions, and the expected traffic volume.

It may on occasion be necessary for the originator to specify that a message should not be fragmented. It may also be necessary to indicate that a recipient MTA is not capable of reassembling messages.

Messages entering a non-fragmenting network from a fragmenting network must be stored in their entirety at the gateway node. This implies that such gateways must have extra storage capacity.

A system might be designed to handle several classes of message sizes. A combination of schemes might prove useful; *e.g.*, one technique for "small" messages and a separate technique for "large" messages. This is possible since the size of a message is known before it is sent. Note that the introduction of the capability to efficiently transfer large messages (*e.g.*, by the implementation of a file transfer utility) may itself affect network traffic characteristics.

3.2.3 Layering Concerns

There is some question as to the right networking level to perform fragmentation and reassembly. The application program, UA, or MTA could perform the function.

If the maximum message size is predetermined, then fragmentation and reassembly could be performed by an application program or UA since no size negotiation needs to be performed. A message could be broken at the source into as many fragments as necessary. Fragments would never themselves be fragmented. This network-wide fixed size approach

is simple but has the disadvantages described earlier.

If the fragment size is not predetermined, then the application program or UA must be able to communicate with its peer on each MTA to negotiate message size and reserve space. This has the undesirable implication that the application must know the routing algorithm used by the underlying message transport layer. It also introduces extra overhead, expense, and complication since arrangements must be made between each pair of adjacent MTAs along the route, using the message system or separate interprocess communication, to negotiate message size. Complications can arise when buffer space is reserved but the transfer is delayed or cancelled. Fragments must also be passed "up" and "down" through an extra protocol layer on each intermediate MTA since communication is between applications.

There are several advantages to having the MTA perform fragmentation and reassembly rather than the application program or UA:

1. A sending MTA may negotiate the message size with a receiving MTA much more efficiently at session establishment time.

2. Providing this service within the MTA provides transparent fragmentation and reassembly to *all* service users. The term *generalized fragmentation* is used for fragmentation performed at the MTA layer.

3. Arguing on design grounds, the service user should not be concerned with this problem. It should submit the message in a single transaction. Higher layer protocols should leave buffering concerns, *etc.* to lower layers which probably already have these capabilities. Concentrating the solutions to these problems in a single layer is more efficient in terms of the amount of code that must be written and execution

efficiency. It allows a more understandable centralized message handling policy to be used. Also, applications should be isolated whenever possible from routing concerns.

A potential disadvantage of performing fragmentation and reassembly at the MTA layer is that X.400-based systems would be forced to deviate from the X.400 recommendations; X.400 does not explicitly provide for negotiation or fragmentation capabilities within the MTA. It would be possible, however, to interconnect an X.400 system that fragments with one that does not by having a gateway perform reassembly before forwarding the message to the non-fragmenting network or by simply suppressing fragmentation on messages destined for a non-fragmenting network (necessitating a maximum message size and separate transit buffer system).

It is possible to support fragmentation at both levels by allowing an application to specify that no fragmentation may occur. The application would receive a non-delivery report message should a message be undeliverable.

3.2.4 Reassembly and Gateways

It may be desirable to combine or *reassemble* two or more fragments being sent between adjacent MTAs. MTAs could reassemble fragments into a larger fragment only taking into consideration the receiving MTA's space situation. Alternatively, they could reassemble taking into account whether the message will pass through low capacity MTAs to avoid having another MTA refragment the message later. The latter scheme would require a distributed data base of "MTA capacities". Moreover, the route a fragment will take may be unpredictable.

Just as a lower level fragmentation strategy must be concerned with the interconnection of networks, so must an application level fragmentation strategy. Depending on the nature

of the adjacent network, a gateway MTA may have to wait for the entire message to arrive before forwarding it to the next MTA. Such characteristics as the maximum message length and whether the adjacent network can reassemble the message must be taken into consideration.

The designers of the ARPA internet protocol chose not to have gateways perform reassembly since in their environment it can lead to buffering problems, deadlock, increased transmission delay, and the necessity for all fragments to pass through the same gateway [Cerf74, Postel81a]. Also, subsequent gateways may need to refragment the packet.

3.3 Deadlock

Designers of S/F networks must take into account the possibility of S/F deadlock. Two types of deadlock are of primary concern [Günther81]. Direct S/F deadlock occurs when two adjacent hosts are both filled up with messages waiting to be transmitted to each other. Messages cannot be transmitted since there are no empty buffers available at either host and none of the buffers can be emptied. Indirect S/F deadlock occurs when there are more than two hosts involved, each waiting upon an adjacent host to accept a message but none with free buffers. This second type of deadlock is possible even when direct S/F deadlock is not [Günther81]. A third type of deadlock, reassembly deadlock, occurs when portions of two or more messages have arrived at a host leaving no free buffer space. None of the messages can be forwarded or delivered since they are incomplete but the missing portion of any message can't be received since there is no place to put it.

Deadlock can be handled in several ways [Isloor80]. Deadlock prevention makes deadlock impossible by preallocation and careful management of resources. Deadlock detection [*e.g.,* Chandy83, Cidon86a, Cidon86b, Chan87] involves the use of a distributed algorithm

that detects the presence of a deadlock situation among a *knot* [Holt72] of nodes. A deadlock resolution scheme [*e.g.*, Mitchell84] is invoked or manual intervention is required when a deadlock situation is detected. The deadlock detection approach offers higher buffer utilization than the deadlock prevention approach at the expense of extra complexity and communication costs. Deadlock avoidance makes deadlock impossible by insuring, at the time a resource is requested, that granting the request will not lead to deadlock. Of course deadlock can be ignored by the system, relying on operator detection and correction instead. Deadlock detection by an operator may be difficult in a distributed environment, however.

A S/F deadlock detection or prevention algorithm is needed to keep MTAs from waiting indefinitely due to running out of buffer space. Because of the goals set out and the nature of the message handling environment, several complicating factors arise. For example, the route a message will take may not be known in advance, the topology of the network or even the number of MTAs in the network may not be known by all MTAs, and a message may be addressed to multiple recipients.

It is still desirable to prevent deadlock even if old messages are discarded, thereby preempting potential deadlock. Since messages may take a long time to traverse the network a "pre-deadlock" lockup state could last a considerable time and a sequence of lockup states may result.

Chapter 4 deals with the problem of deadlock in greater detail and explores techniques apropos to the message handling environment.

3.4 Flow Control

Flow control is an end-to-end mechanism by which a receiver throttles a sender to prevent

data from arriving at a faster rate than the receiver can handle [Tanenbaum81]. *Congestion control* deals with the global problem of data arriving at nodes faster than it can be forwarded [Gerla80, Pouzin81]. Both types of control are necessary for a network to perform well when it is under heavy load.

There are two basic techniques for achieving transport level flow control. Some transport level protocols use a variable-sized window scheme to prevent a transmitter from overrunning a slower receiver. In these schemes the receiver sends a negative acknowledgement to the transmitter or simply does not send an acknowledgement, causing the transmitter to wait. Credit schemes [Pouzin81, Tanenbaum81], on the other hand, prevent the transmitter from sending until it has obtained an indication from the receiver of the amount of traffic it is willing to accept.

End-to-end flow control serves two main purposes. First, it can limit the amount of buffer space concurrently in use at intermediate nodes by an originator-recipient pair, thereby restricting the buffer space used by any connection. Second, flow control can prevent the originator from continuing to transmit when the recipient or any intermediate node is not accepting packets.

3.5 Congestion Control

Figure 3.1 compares the throughput of a theoretical or idealized network to that obtained by both controlled and uncontrolled network load in a real network. Each node in the idealized network always has complete knowledge of the network state and there is no control mechanism overhead. Throughput remains constant while delay increases after the idealized network reaches the saturation point. If a network imposes no control over message traffic, throughput will typically follow the curve for the idealized system but begin

to drop after a certain point and decline to zero if the system becomes deadlocked. Using a suitable control mechanism, throughput can increase monotonically until the saturation point is reached, after which throughput remains stable. The network does not achieve the maximum theoretical throughput because of the overhead of the mechanism and reduced resource sharing. Naturally, different control mechanisms will offer different degrees of performance. If maximal throughput performance is desired, it is the network designer's goal to develop a mechanism having a throughput curve that closely tracks the theoretical throughput performance.

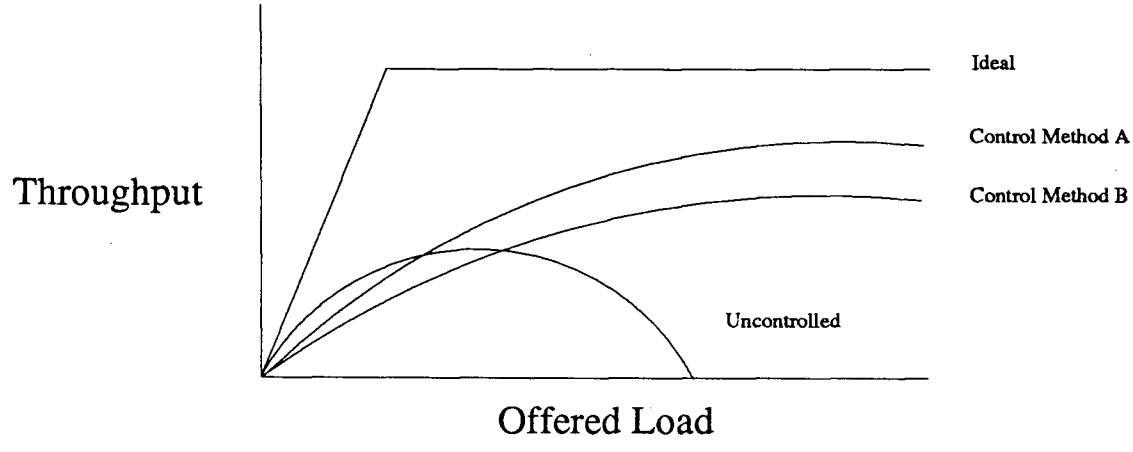

Figure 3.1: Network Throughput vs. Offered Load

Because the ideal throughput curve is a result of complete global knowledge of the network state, it seems obvious that by distributing network state information to each host higher network throughput could be realized. For example, information about failures, buffer space utilization, or RTS connection usage could be distributed. Distribution of this

information, however, requires use of resources (*e.g.,* processing overhead, buffer space overhead, network bandwidth requirements) and adds another element of complexity to the congestion control mechanism. Furthermore, unless some types of information can be distributed in a timely manner it may be of limited use or even invalid by the time it is delivered to the recipients. There are also potential problems associated with different hosts seeing different versions of the information at the same point in time.

A compromise can be reached by distributing certain types of long-lived information that can be used to improve general performance. It does seem undesirable to globally distribute large amounts of network topology, short term statistics, and details about communication between adjacent clusters because of the expense and volatility of the information.[6] Network subsystems such as message routing and deadlock prevention may require distribution of their own types of information and so distribution of information useful to a congestion control subsystem may involve little additional performance degradation.

Congestion results when the demand for certain network resources is higher than the supply. The resources of primary concern are buffer space and RTS connections, although gateways in particular may experience high CPU demands. Traffic begins to back up when the arrival rate of new messages exceeds the message throughput rate. The problem can be exacerbated by network heterogeneity and statistical variation. Some parts of a network can be congested while others are not. Also, the message arrival rate is typically not constant, but varies considerably over the course of a day, week, month, and year. Likewise, the theoretical (optimal) throughput can vary over time due to failures or changes in network topology. Application level congestion can be increased by congestion in underlying network layers and delays in acceptance of messages by recipients (*e.g.,* because of lack of

[6] Although the UUCP internetwork, a widely used, unreliable message transport facility that uses source routing, has been doing just this for years.

buffer space or unreceived message fragments).

The principle problem addressed by a practical congestion control scheme is the distributed management of resources with limited knowledge of global state information and even less information of the future state. A congestion control scheme can be placed somewhere on a "liberal" – "conservative" continuum, where a liberal scheme is quick to allocate resources to try to optimize throughput over the short term and a conservative scheme tries to maintain a high continuous throughput.[7] A liberal scheme may perform badly when it encounters unexpected changes while the conservative scheme will, under normal conditions, perform less well than a liberal scheme but much better under adverse conditions. Again, this is because the conservative approach makes less efficient use of network resources when the network is not heavily loaded. The performance of a scheme is dependent on the current and future state of the network and this information is in general unobtainable.

How can a liberal scheme continue to provide acceptable throughput when its assumptions no longer hold? One alternative, used in packet-switching networks, is simply to discard messages. Apart from the significant expense of doing this for application level messages, the problem is shifted to a decision about which messages should be discarded to guarantee reduced congestion while being fair to network users. If messages cannot be discarded, rerouting to archival hosts (*i.e.*, specially designated hosts equipped with large amounts of secondary storage to temporarily store messages) could be useful, although this would tend to increase congestion around archival hosts.

An adaptive scheme that provides adequate performance over the entire range of traffic load seems to be indicated. The design should not restrict traffic unnecessarily under

[7] There is a similarity here with deadlock prevention versus deadlock detection and resolution techniques. Deadlock prevention is more conservative than deadlock detection and resolution.

light congestion and should provide reasonable performance under heavy load and various failures. In the design of such a scheme there will necessarily be tradeoffs between complexity, resource requirements, and performance. Given a large degree of freedom in network topology and the possibility of failures, any design for handling flow and congestion control is not likely to be always "optimal"but may provide adequate performance over all (or at least many) topologies under various conditions.

3.5.1 Network and Transport Level Congestion Control

Many techniques have been proposed for congestion control in the packet-switching environment, but few have found their way into "real" networks, perhaps in part due to assumptions made to simplify analysis. Much research has addressed flow and congestion control in virtual circuit oriented networks. In this section several approaches to congestion control are described for the network and transport levels. The suitability of these approaches to the message handling environment is discussed in the next section.

Congestion control could, in theory, be achieved by directly adjusting the rate of input to the network (*e.g.*, a particular node can submit up to N bytes per second) [Cerf81]. The NETBLT transport protocol [Clark87] uses rate flow control to match the data transmission and consumption rates in packet switched networks. This is difficult to realize for a network of arbitrary topology, however. It is hard to determine what this rate should be for each node and the rate will be affected by changes in the topology.

In the packet-switching environment, congestion can be prevented by placing a limit on the total number of packets in the network. In an *isarithmic* network [Davies72], this is achieved by holding the number of packets in the network constant. This approach has several difficulties, including prevention of localized congestion.

Kleinrock and Tseng [Kleinrock80] studied a class of flow control techniques based on limiting the permit generation rates of logical channels. An interesting element of the research is that the throughput performance curve can be brought as close to the ideal curve as desired simply by increasing the buffer size allocated to each physical channel.

A common way for a node to deal with the arrival of too many packets or datagrams is simply to drop them. Another method involves having congested intermediate nodes set a bit in the header of a packet, allowing all nodes along the packet's route to take appropriate steps. A third alternative is to send a *choke packet* [Tanenbaum81] to the source which includes the destination of the packet involved in the congestion. When the source receives a choke packet it reduces its transmission rate to the destination. The Internet Control Message Protocol [Postel81b] defines a *source quench message* which may be sent by a gateway or destination host to the source host to reduce the rate at which datagrams are being sent to an internet destination. The source quench message includes part of the datagram involved in the congestion so that the source host can associate the source quench message with the appropriate process. In both of these schemes regulation of the flow is not likely to be smooth [Mills87]. Also, the message sent back to the source may itself be dropped and there is nothing to prevent the source from ignoring a source quench message [Nagle84].

Datagrams have been traditionally defined as single submission event entities, unrelated to any subsequent submission. The notion of a *flow* has been proposed [Zhang87] where sequences of datagrams from a particular originator to a particular recipient share a *flow identifier* that uniquely identifies the sequence at the datagram level, allowing end-to-end flow control. Zhang [Zhang87] proposes a packet-switching network architecture where the user provides a service specification that expresses resource demands such as the user's

expected average transmission rate.

Congestion control based on *input buffer limits* [Lam79, Lam80, Lam81] controls the network input rate by discriminating against input traffic in favour of transit traffic and imposing a limit on the number of buffers input traffic can occupy. These techniques do not directly change the rate of input to the network. When the network is congested the input buffers do not help with the transport of messages; they tend to hold submitted messages for a long time since they can not be forwarded. These buffers would be used more effectively as transit buffers.

A dynamic input buffer limit technique can adjust to the congestion by decreasing the number of poorly utilized buffers by shifting them to other uses. On the other hand, reducing the number of input buffers too much may cause a decrease in the network throughput since some outgoing links may not be fully utilized. The difficulty is attaining balanced buffer utilization to maintain a sustained high throughput rate.

Lam and Reiser [Lam79] discovered a capacity law that can be used to determine the input buffer limit of a node, assuming a fixed network input pattern and fixed routing. This capacity law states that the input buffer limit B_I/B_T must be less than the ratio of input message throughput to total message throughput σ_T, where B_I is the maximum number of buffers that may be used by input messages and B_T is the total number of buffers in the node. Schwartz and Saad [Schwartz79] discuss this further and propose a slightly different mechanism. Lam and Lien [Lam81] extend the determination of input buffer limits to heterogeneous networks by multiplying σ_T by a constant, $\alpha < 1$. Smaller values of α allow the network to withstand larger fluctuations in traffic load over longer intervals.

Routing mechanisms can be designed to keep track of delays along various paths in an

attempt to provide minimum delays. This is an area where the study of congestion control and routing overlap. Rerouting of packets around areas of congestion has been considered, although it is possible for this action to actually spread the congestion. Unless a node has enough global information it is difficult for it to decide whether rerouting will help.

3.5.1.1 Buffer Class-Based Congestion Control

Raubold and Hänle [Raubold76] introduced the deadlock-free *structured buffer pool* which separates the buffers within each node into an ordered set of *buffer classes.* At each hop along its route a message uses buffers belonging to strictly increasing buffer classes. A two level (node-to-node and end-to-end) dynamic window mechanism is used to provide flow control. Each virtual channel is assigned to a certain buffer class at each node along its path at connection setup time. Several different approaches have been taken concerning the problem of how to organize the structured buffer pool and move messages through the buffer system.

Flow and congestion control for structured buffer pool based transport systems were first studied by Giessler *et al.* [Giessler78, Giessler81] for the virtual circuit based GMD network. The input buffer limit technique is generalized by dynamically adjusting the number of buffers belonging to each buffer class based on traffic conditions. They define the goals of *global flow control* ("G-control") to be the prevention of global congestion that causes subnet throughput degradation, the economical utilization of network resources, and the fair treatment of different traffic groups (input, transit, and output groups). Congestion control is performed in a fully distributed and adaptive manner. In simulation studies, the combination of individual (per virtual circuit) flow control ("I-control") and global congestion control provided satisfactory network behaviour when the network's saturation

point was reached. An analysis of the structured buffer pool when the communication channel is not the bottleneck has also been performed [Wunderlich80].

Any scheme that adjusts buffer class limits dynamically cannot actually achieve a reduced buffer class limit until the buffer class usage falls to the new limit. If the message transmission rate of a buffer class marked for reduction is high then this is not likely to be problematic, since buffers would be freed frequently and subsequently reassigned to a new buffer class. If the transmission rate is low there may be a considerable delay before the new buffer class limits are met.

A problem with G-control is that it is not a fair policy: traffic is held back based solely on how far it has travelled rather than individual sources of the congestion being controlled. For example, a decrease in buffer space available to messages that have travelled N hops to get to a particular node may be made. This will affect all messages that have travelled $N-1$ hops that need to be forwarded to this node (and, recursively, to those messages that have travelled $N-2$ hops that need to be forwarded to those nodes, *etc.*), not taking into account such factors as where messages are coming from (both in a local and a global sense) or whether the originator of a message is using "too much" of the network's resources. Fairness is increased through the combined use of I-control since the end-to-end window size can be adjusted to the actual acceptance rate.

3.5.2 Application Level Flow and Congestion Control

While the goals of flow control and congestion control for the message handling environment are similar to those at other networking levels, realization of these goals can be more difficult. This is a result of the high degree of freedom allowed in the network topology and the corresponding model of S/F transfer. In general, the high degree of network

heterogeneity makes real time distribution of network performance data impossible. On the other hand, the offline nature of the data transfer as well as the more detailed information contained in the message envelope open some avenues of investigation not possible in other environments. In packet-switching networks the emphasis tends to be on low delay and the subnet supports rapid distribution of network performance data. In contrast, the emphasis in the message handling environment tends to be on high message throughput while the subnet may not lend itself to rapid dissemination of performance data.

The notion of flow control at the application level in the message handling environment most closely resembles datagram level flow control as both submit data for transmission in a single operation. There have been proposals for datagram level flow control [Elie79, Zhang87] but they have yet to become widely accepted, perhaps because independent routing is considered to be an important characteristic of datagrams. Also, because there may not be an originator-to-recipient connection, end-to-end flow control can not be performed in the same manner at the application level as for topologies where links are normally continuously available. Thus, in contrast to a virtual circuit, there is no end-to-end feedback (*e.g.*, in the form of acknowledgements) to use for flow control. Given no means of performing end-to-end flow control, flow control at this level is largely limited to restricting the rate of network input so that resources are shared in a more equitable manner; *i.e.*, the flow of individual messages from a particular user, system, or cluster may be regulated.

There are several problems unique to, or more severe in, the message handling environment. These include:

- As the normal mode of operation, RTS connections may not be available continuously. This is likely to contribute to problems with availability and sharing of buffer space.
- Heterogeneity causes problems because there are likely to be many bottlenecks through-

out the network. The amount of resources (buffer space, link capacity, *etc.*) available will vary from cluster to cluster and from host to host. Bottlenecks and failures contribute to traffic backing up, leading to congestion.

- A significant component of congestion is due to the long lifetime of messages within the message handling system and the need to avoid discarding messages. When unaccepted messages consume all of the buffer space at a recipient MTA messages begin to "back up" and occupy transit buffers in upstream MTAs. Messages may not be accepted because an MTA is down or simply because a user does not accept delivery of incoming messages.

- Unlike most other environments, messages may arrive "offline" and may come from any source. Furthermore, the recipient need not have solicited the message nor be active when the message arrives. This has important consequences regarding managing received messages.

Due to the offline nature of message transfer (*i.e.*, there is no real-time requirement), there can be more network control over admitting new transfers to the network. Often it does not matter to the originator of the message whether it is delivered within one hour or overnight. A flow control scheme could use this characteristic to distribute message traffic more evenly over time, perhaps by coordinating transfers between interior MTAs.

The potentially long lifetime of messages within the network, coupled with a prohibition on discarding messages, implies that the number of messages in each traffic group using transit buffers must be limited over the entire network. This strongly suggests an end-to-end mechanism to insure that messages can be accepted by a recipient before the message is sent and that buffer space is allocated fairly by the recipient to arriving traffic

groups. In addition to this end-to-end mechanism, a second mechanism is required to prevent congestion within the S/F component of the network and to insure fair allocation of resources.

If the most important uses of the network are seen to involve the transfer of smaller messages (*e.g.*, interpersonal messaging, queries, *etc.*), then steps must be taken in the design of the network to see that large transfers do not unfairly impede smaller ones. Large transfers have much higher demands for buffer space and RTS connection time. Of primary importance is that the network should avoid allocating significant quantities of transit buffer space to a large transfer. Apart from fairness considerations, a stalled transfer uses transit buffer space that could be better used by more "successful" transfers.

One solution to buffer space contention problems between large and small transfers is to provide separate buffer pools for the two classes of traffic. To avoid the situation where one pool has free space while the other is full, each buffer pool could be given a minimum allocation and a third, shared buffer pool could be established. The number of buffers available to each pool could be dynamically adjusted based on traffic conditions.

Congestion in underlying layers may occur when an RTS connection uses a shared medium. While there may be some value in higher level congestion control mechanisms cooperating with lower level mechanisms [Herrmann76, Gerla80], this possibility will not be explored here.

3.6 Fairness

It is difficult to precisely define the concept of fairness in a computer network, particularly for heterogeneous networks. It is much easier to *identify* unfair treatment of a particular message or class of messages. For example, the network topology can be such that a greater

than normal share of some resources are made available to transfers between certain pairs of hosts or some hosts may have a lower probability of establishing a virtual circuit with a particular set of hosts when the network is heavily loaded. The ultimate form of unfairness is *livelock*,[8] where certain messages can never be forwarded because of routing, scheduling, buffer management, or deadlock prevention policies, for example.

Components of network resource management, such as flow control, congestion control, buffer management, and scheduling are largely concerned with managing shared resources and therefore must take fairness issues into account. Rudin defines flow control in a global, fairness-oriented way as "that collection of algorithms which are used in a network to prevent a single user or single user group from hoarding the resources of the network to the detriment of others" [Rudin76]. In the end, it is up to the network designers and administrators to arrive at an operational definition of fairness and to decide how much they are willing to pay for it. At the very least all hosts and clusters should receive some minimum level of service.

There is an intuitive, cause and effect component to fairness. A particular user, host, or cluster should not be able to capture an excessive share of network resources. For an individual connection this is partially accomplished by *selective flow control* performed by a virtual circuit's end-to-end flow control mechanism. Filling up the circuit's window causes the network to refuse further input from the originator until some of the data in the circuit has been accepted by the recipient. It is the user of the circuit that is held back by the flow control mechanism, not other network users. Selective flow control does not in itself make the network fair. For example, there may not be any direct network control over the duration of a connection or the number of connections simultaneously in use by a

[8] Günther [Günther81] differentiates between two forms of livelock, *starvation in poverty* and *starvation in wealth*.

particular user, host, or cluster. There is usually less of this cause and effect relationship in datagram networks because datagrams are transported independently: end-to-end flow control must be performed at a higher networking level if it is present at all. Fair use of buffers is harder to achieve and poor performance results when the user or users responsible are not regulated [Nagle87].

Proportional access to network resources can be specified by providing a *definition of fairness* that specifies the ratios of resources to be allocated to traffic groups. Messages within traffic groups are treated equally but some traffic groups are given preferential treatment. Such a specification could preclude livelock by insuring that traffic groups having a higher service priority are prevented from locking out lower priority traffic groups for too long. An example of this arises in the treatment of message priorities. If high priority messages are always transferred ahead of lower priority messages it is possible for a continuous flow of high priority messages to lock out all other messages. If there are three priority classes, "high", "normal", and "low", "high" could be assigned a numeric priority twice that of "normal", and "normal" twice that of "low". Messages might then be multiplexed such that for every low priority message two normal priority messages and four high priority messages would be forwarded.

There is comparatively little literature primarily concerned with fairness issues in S/F networks. Jaffe [Jaffe80] discusses fairness properties of a policy that achieves an ideal delay-throughput tradeoff in a fixed route, virtual circuit environment. Bharath-Kumar and Jaffe [Bharath-Kumar81] note that schemes that optimize a global performance measure must be careful not to force some controlling parameters to zero, thereby creating unfairness. An example of this is given in Figure 3.2 where messages are to be sent from Host 0 to Host 0′, Host 1 to Host 1′, ..., Host N to Host N'. The global throughput is

maximized by suppressing all traffic between Host 0 and Host 0′ since any traffic between these two reduces the global throughput.

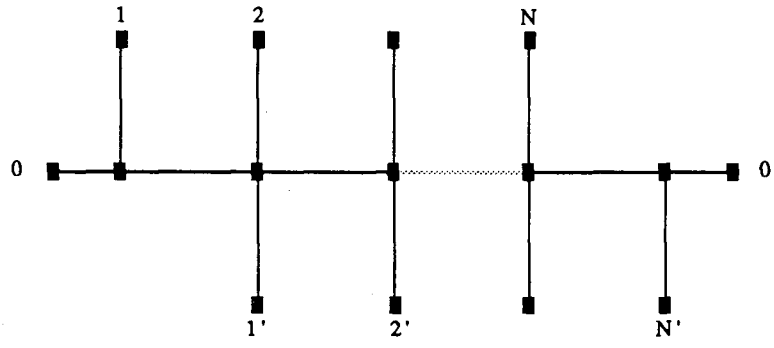

Figure 3.2: Unfairness Through Maximized Global Throughput

Traffic is commonly divided into two types when designing congestion control mechanisms: input traffic (traffic waiting for admittance to the network) and transit traffic. Fairness in admittance involves the control of access to input buffers on a per user basis as well as control of the number of buffers used for input buffering. Fairness in transit involves equitable use of network resources, such as buffer space and RTS connections, for messages waiting to be forwarded. Unfairness can result when transit traffic monopolizes buffer space, shutting out input traffic.

Round-robin (circular-scan) link multiplexing with window-based flow control has been shown to provide a simple means of fair flow control in a fixed path, virtual circuit environment under heavy demand and with sufficiently large window sizes [Hahne86]. It can also assist in insuring individual users or traffic groups get equal, or at least proportionate, access to resources [Katevenis87]. Scheduling algorithms that attempt to provide equal mean end-to-end delays to each user have been studied [Wong82, Rahnema88].

Simplistic multiplexing without appropriate concomitant buffer management does not insure fair treatment. For example, using a first-in, first-out (FIFO) ordering of messages to determine the order in which messages are allocated buffers and forwarded may not result in fair treatment. Figure 3.3 shows a portion of a network that uses FIFO message multiplexing. Messages are denoted by boxes labelled with their originating host's name. As depicted, a message is being forwarded from Host B to Host D and from Host A to Host E. Assuming network homogeneity (*i.e.*, equal transmission rates, number of buffers, round-robin scheduling, *etc.*), messages transferred from Host A to Host E can acquire a disproportionate quantity of buffer space at the expense of messages from both Host B and Host C transferred to Host E through Host D. This is because Host B and Host C alternate sending messages to Host D, and Host A and Host D alternate sending messages to Host E. For every two messages from Host A that are allocated buffers at Host E, one message from each of Host B and Host C obtain a buffer.

An example serves to illustrate another unfair congestion control policy. A policy could always accept a forwarded message that is addressed to its host (a "preferred" message) before any message that will have to be forwarded. This might appear to be a good strategy since, once transferred, the preferred message is removed from the network, thereby freeing buffer space. It is conceivable that under heavy load, however, some messages will become livelocked since there could always be a preferred message to transfer.

Allocation of RTS connections can also be managed unfairly. Letting large transfers retain sole use of an RTS connection for the duration of the transfer is unfair to small transfers which must wait. Possible partial solutions include multiplexing several RTS connections onto a single transport connection or fragmenting messages into equal size pieces and multiplexing the fragments (*e.g.*, by round-robin scheduling) onto the RTS

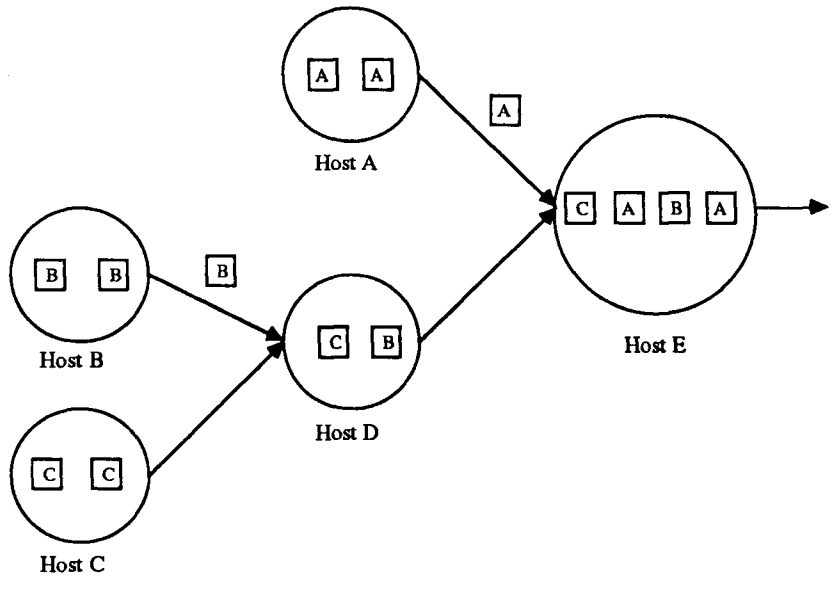

Figure 3.3: FIFO Message Multiplexing

connection.

There is a strong temptation to use available resources rather than to let them remain idle. Because of the bursty nature of network traffic, "overallocation" to one traffic group can lead to subsequent unfair treatment of other traffic groups. The duration of this unfair treatment is related to the rate at which buffers can be released from the overallocated traffic group.

Dynamic resource sharing policies tend to provide increased fairness in proportion to a host's message throughput. At T_1 a host may be treating all traffic groups "fairly" but when a new traffic group is created at T_2 there may be a time lag (due to existing queues and buffer allocations) before the new traffic group can acquire a "fair" share at T_3. In some cases there may be a tradeoff between response time (delay) and fairness in that a "fair" resource allocation is not necessary. Perhaps the new traffic group can be serviced

between T_2 and T_3, albeit with an "unfair" resource allocation. In such cases the overhead in adjusting resource allocations may not be justified. Knowing the resource demands of a particular transfer *a priori* can assist in determining a fair resource allocation.

There are costs associated with fairness, including resource usage and increased complexity of the system. Guaranteeing that certain traffic groups do not monopolize network resources implies that limits be imposed on resource usage. In particular, minimum buffer resources must be dedicated to each traffic group (or at least there must be some reasonable upper bound placed on the delay for buffer availability) so that no traffic group is discriminated against. In this case the cost may be in the form of poorly utilized buffers.

In summary, because fairness is a subjective characteristic, it is difficult to define. Various traffic groups can be identified and may demand different levels of service. An approximation of fairness can be achieved by trying to meet service requirements defined for the network. Some type of end-to-end flow control is required to throttle back messages from originators that are already using their share of resources. There is often a tradeoff between fairness and underutilization of resources. Fairness issues are discussed in Section 5.6 with respect to the framework presented in Chapter 5.

3.7 Network Traffic Patterns

To help with the design of a better message transport system, observations of operational systems should prove useful. These observations center on a categorization of hosts that may make up the system and message traffic.

A *leaf MTA* does not have messages passing through it: all traffic is due to submissions from users and deliveries to them. An MTA may be an *interior MTA* or *gateway MTA* in addition to being a cluster gateway. An interior MTA is an MTA that performs S/F duties

and also services a local user population. A gateway MTA handles much more traffic than the other two classes since, in addition to performing the functions of an interior MTA, it serves as a gateway to one or more networks.

A significant proportion of network traffic has regularity associated with it. For example, distribution lists tend to be static over long periods of time and submissions are often collected into digests and redistributed to a list's members with a predictable periodicity, gateways tend to regularly receive notices, digests, and bundled news for redistribution. It might be possible to take advantage of these regularities, for example in buffer management.

An exception to the normal traffic pattern occurs when messages cannot be transferred between two adjacent MTAs (*e.g.,* because a host is down) and there is no alternate route for messages to take. When the two MTAs once again communicate, a backlog of accumulated message traffic begins to flow, causing a large "wave" of traffic to pass through the network. This sudden increase in the volume of messages tends to create congestion.

Measurements of the EAN messaging system at the University of British Columbia are summarized in Table 3.1. Message size statistics are summarized for host `ubc-ean`, which acts both as a network gateway and a cluster gateway, and host `ubc-cs`, a leaf MTA. *Exported messages* are those transmitted by an MTA and *imported messages* are those received by an MTA. These messages belong primarily to the CDN, UUCP, CSNET, ARPA, and BITNET domains. Message traffic at the `ubc-ean` gateway is distributed equally over the entire day. In a "well connected" leaf MTA such as `ubc-cs`, traffic is distributed evenly over working hours while those with less frequently available RTS connections tend to experience burstier traffic. Most messages are under 3000 bytes long but there are many much larger messages, some more than one megabyte in length. Recent measurements show that traffic through the CDNnet gateway has increased to about 5500 messages per

day, totalling approximately 15 megabytes [CDNnet88].

Host	Interval	Messages	Message Sizes (bytes)			
			Max.	Mean	Median	% < 3000
		Imports				
ubc-ean	30/9/86 - 9/10/86	4,904	1,586,437	7,872	779	84.9
ubc-ean	30/9/87 - 9/10/87	8,733	154,139	3,962	954	83.7
ubc-cs	7/10/86 - 12/10/86	484	542,597	16,065	1,117	78.5
ubc-cs	28/11/87 - 3/12/87	647	201,912	2,263	907	89.8
		Exports				
ubc-ean	30/9/86 - 9/10/86	7,916	1,586,437	4,688	935	83.2
ubc-ean	30/9/87 - 9/10/87	17,414	154,139	3,224	1,219	84.1
ubc-cs	7/10/86 - 12/10/86	207	35,469	1,562	640	93.7
ubc-cs	28/11/87 - 3/12/87	274	2,783,412	22,761	500	94.2

Table 3.1: Message Traffic Summary

The study of message traffic clearly indicates two classes of traffic: relatively small messages and much larger messages. The EAN messaging system provides no file transfer facility. The introduction of such a capability is likely to increase both the volume and the mean size of larger messages. These observations will be taken into consideration in the design of the message transport system in Section 5.

3.8 Summary

Message fragmentation has been proposed to permit large messages to pass through the network as well as allowing for improved performance in a S/F message-based system. Schemes developed for flow control, deadlock prevention, and buffer management in packet-switching networks have been described and their suitability for use in the message handling environment has been discussed. Alternative schemes for buffer management in the message handling environment have been outlined. Issues associated with fair treatment of messages have been considered.

Chapter 4

Store-and-Forward Deadlock Prevention

Most research into methods of preventing store-and-forward deadlock in computer networks has been focused on the network level. However, many of the techniques for handling deadlock at the network level are unsuitable for use at the application level. This chapter addresses the problem of application level deadlock prevention and introduces a hierarchical scheme for the message handling environment that combines the structured buffer pool method as a top level with appropriate deadlock-free bottom level transport mechanisms.[1] The natural composition of a network from interconnected groups of hosts makes it well-suited to combining several deadlock prevention techniques. The advantages of the selected schemes are retained while reducing the cost of their disadvantages, creating a more practical system.

The rest of this chapter is organized as follows. Section 4.1 outlines the problems with existing deadlock prevention schemes in this environment. The hierarchical solution together with a proof of correctness is presented in Section 4.2, and Section 4.3 concludes the chapter.

[1] This material also appears in [Brachman89c].

4.1 Existing Techniques

Raubold and Hänle [Raubold76] introduced the deadlock-free structured buffer pool that partitions the buffers within each host of a store-and-forward network into an ordered set of buffer classes. Merlin and Schweitzer [Merlin80] showed that S/F deadlock implies the existence of a cycle of buffer requests in the buffer graph representing the network; a loop-free buffer graph implies that deadlock cannot occur. The structured buffer pool prevents deadlock by allowing a message to use only buffers belonging to strictly increasing buffer classes at each transfer. An advantage of the buffer class approach to deadlock prevention is that it can be integrated with flow control and congestion control mechanisms. Such integration allows the network designer to address the flow control and congestion control tasks without concern of introducing direct or indirect deadlock, subject to obeying the rules governing the structured buffer pool. Apart from placing an upper bound on the number of hops a message can take to reach its recipient, the buffer class approach imposes no constraints on the routing of messages.

Several different approaches have been taken concerning the problem of how to organize the structured buffer pool and move messages through the buffer system. Toueg and Ullman [Toueg79] and Toeug [Toueg80] have developed schemes based on the structured buffer pool technique. Merlin and Schweitzer [Merlin80] and Günther [Günther81] have developed efficient methods of preventing S/F deadlock based on preventing cycles of buffer requests. Their respective "valley counting" and "counting peaks and valleys" schemes, both based on buffer classes, can be adapted to the MHE by using the unique MTA identifiers to represent the node numbers and choosing a "sufficiently large" estimate of the maximum hop count in the network. This is unsatisfactory, however, since the maximum hop count is unlikely to be static and an excessively high choice results in low buffer utilization.

Wimmer [Wimmer84] has developed a scheme, based on Günther's work, that allows a tradeoff between routing constraints and the number of buffer classes required.

Gopal [Gopal85] has derived a structuring scheme that often requires significantly fewer buffer classes than other methods. Another scheme [Bongiovanni87] minimizes the number of buffers for certain specific network topologies.

A significant problem with the use of the structured buffer pool technique is that the number of buffer classes tends to be large. At least one buffer for each buffer class must be permanently allocated at every MTA on the network. In the worst case, the number of buffer classes required is the maximum number of MTAs a message can pass through as it travels from the originator to the recipient MTA. Much of the time this is wasteful, since a message following such a path is probably rare. Using the structured buffer pool technique at the application level (such as in the message handling environment) is unattractive if a large number of buffer classes are required since buffers at this level are much larger than at the data link or network layers.

Several methods based on the structured buffer pool have been described for combining deadlock-free subnets into a deadlock-free interconnection of the subnets, although they do not address the problem of reducing the number of buffer classes [Merlin80, Günther81]. None addresses the issue that a change in the topology may require changes at distant locations in the network. While reducing the number of buffer classes required, Gopal's method [Gopal85] still has the drawback that a change in the topology may require changes at distant locations in the network. It also involves either the network administration or a distributed algorithm to form a partitioning of hosts to determine the number of buffer classes required.

Gelernter [Gelernter81] proposes a scheme that reroutes packets around areas of poten-

tial deadlock and requires one arbitrarily chosen node to accept any packet within a finite time, even if this requires dropping a packet. The topology of the network is required by the algorithm but it imposes no routing restrictions, does not require that the buffer pool on each node grow with the network size, and does not require buffer pool partitioning.

Buffer management schemes that require preallocation of space are deadlock free. For example, a virtual circuit setup operation may reserve transit buffers for the lifetime of the circuit at each hop along the route. This approach has the advantage over buffer class schemes that no special action needs to be taken when the network configuration changes. It has the disadvantages that there is extra overhead in preallocating space and a special mechanism may be needed to free resources that are no longer required.

Gambosi *et al.* [Gambosi84] have developed a distributed deadlock detection and recovery algorithm. The scheme incurs no overhead for sites not involved in the deadlock so that normal network traffic is not affected by the deadlock detection traffic.

A scheme using circulating tokens has been proposed by Konorski [Konorski86]. Nodes cooperate via an inquiry-response procedure to determine whether an incoming packet should be accepted or rejected.

Another solution to deadlock, the *time-stamp technique* [Błażewicz87a, Błażewicz87b] orders messages by assigning a unique time stamp to each one.[2] Upon creation, a message is given a time stamp that identifies where and when the message was created. A comparison of the time stamps of any two messages in the network will identify one message as the "younger" and the other as the "older". Forwarding of messages proceeds without restriction until a sending host is unable to forward a message to a receiving host because

[2] This method can be viewed as a generalization and refinement of the overflow handling technique described in [Kahn72].

the receiver has no free buffer space. If the sender's message is older than some message at the receiver, then the two messages are exchanged using a special exchange buffer, otherwise no transfer takes place. The algorithm guarantees that even if no other message can be transmitted, the oldest message in the network will eventually be delivered. The method has several desirable properties, including network size and topology independence, freedom from livelock, and not imposing any restriction on message routing.

Simulation of the NPL network [Giessler78] shows that that network may experience deadlock immediately after the traffic load exceeds a value called the *critical applied load.* On the other hand, there are reports of networks running for an extended period of time without experiencing deadlock [Gerla81]. The likelihood of deadlock depends on the network topology, network traffic, buffer management, and the routing algorithm used [Kahn72]. It is expected that in a heterogeneous network, where hosts may have various numbers of links of varying capacity and availability, deadlock is more probable.

Measurements of the EAN messaging system (presented in Section 3.7) show that most messages are under 3000 bytes long but there are many much larger messages, some more than one megabyte in length. The proportion of large messages is expected to increase as more applications are based on message transfer systems. Because of the relatively high cost of moving messages, techniques requiring the deletion of messages are not attractive. Deadlock detection and resolution techniques are not likely to be effective since the MTS model makes no assumptions about the topology of the network, including availability of communication links. In general, deadlock prevention is the best way to deal with the deadlock problem in the messaging environment.

The use of a "pure" (*i.e.*, network wide) structured buffer pool technique is impractical in the message handling environment since the topology is subject to frequent change and

individual hosts typically have limited storage. The number of buffer classes needed by each MTA on the network may be increased by the addition of a new MTA at some "distant" location on the network. Besides being wasteful of buffer space, this increases the amount of administrative work that needs to be performed. In cases where there is no centralized management (*e.g.*, Usenet), this is difficult or impossible.

It is desirable for a deadlock prevention algorithm to depend as much as possible only on the configuration of a cluster (*i.e.*, readily available information). Adding a new MTA to the network or changing the connectivity of the network should not require changes to the number of reserved buffers at MTAs distant from the change. Potentially confidential information, such as the number of hosts within each cluster, should not be required by the algorithm. Of course, any scheme should also be easy to understand and for network administrators to maintain. Current buffer class techniques lack these properties.

Using the structured buffer pool technique only between clusters reduces the number of buffer classes required since now a hop means a transfer between clusters and is independent of the number of transfers that occur within clusters as a message is forwarded to its recipient MTA. Changes to the number of clusters in the network are usually far less frequent than changes to the number of hosts in the network. Therefore, restricting use of the structured buffer pool to intercluster traffic means that the frequency of changes to the number of buffer classes will be greatly reduced over a pure structured buffer pool approach.

With respect to the time-stamp technique, a drawback is that younger messages may be displaced from their preferred route in favour of older messages. This may imply increased communication costs and significant delays when some network configurations are under heavy load. In particular, a younger message may be displaced across a slow speed link or

a link that is expensive to use. The severity of this problem would be expected to increase as the connectivity of the network decreases.

Howard [Howard73] describes how an operating system hierarchically organized into layers can use a mixture of several deadlock control policies, one policy per layer. The proposed solution, described in the next section, applies this approach to the S/F network by creating a two-level hierarchy.

4.2 A Hierarchical Solution

The natural composition of a network from interconnected clusters of hosts offers a clear two-level hierarchical structure for message flow: intracluster and intercluster flow. The proposed method uses the structured buffer pool technique for the top layer to prevent deadlock among clusters and another technique or combination of techniques as the bottom layer to provide deadlock-free message transport within individual clusters. A particular cluster is free to choose the technique to be used for its cluster layer. This scheme allows changes in network topology within a cluster to be isolated from all other clusters and is important for cluster autonomy.

The use of a structured buffer pool technique to join clusters of MTAs is a natural choice since, in addition to having the property that messages always flow down their preferred route (*i.e.*, the algorithm does not require the rerouting of messages to prevent deadlock), it can be used as a means of flow control and congestion control between clusters. The time-stamp approach does not have these desirable properties. Any suitable technique may be chosen for organizing the structured buffer pool, including methods that reduce the number of buffer classes required over the simple hop-count method. The particular technique chosen, together with the number of clusters and, for some techniques, the net-

work topology, determines the number of buffer classes required. An appropriate choice for the deadlock prevention technique used within a cluster of hosts depends on the network topology of the cluster. The hierarchical technique can significantly reduce both the total number of reserved buffers and the number of buffer classes over the entire network compared to pure buffer class methods.

It might seem that an "obvious" approach would be to construct a two-level structured buffer pool system by interconnecting individual systems, each managing a cluster, to a global system that ties the clusters together. Messages could be moved among the intracluster and intercluster structured buffer pool systems by reentering either system at buffer class zero. Because this straightforward approach introduces a cycle of dependencies in the graph representing the buffer system, it fails to solve the problem since deadlock is still possible. This is illustrated by the following example.

Consider the two-level system shown in Figure 4.1 where there are two clusters, each containing three MTAs.[3] A corresponding two-level structured buffer pool is shown in Figure 4.2.

The structured buffer pools depicted in Figure 4.2 consist of three buffer classes (BC0, BC1, and BC2) for each member of both intracluster buffer pools, each buffer class having a single buffer. MTAs C and D, which are cluster gateways (*i.e.,* they contain either end of the link connecting the two clusters), also contain the two buffer class intercluster buffer pool. Messages in buffer classes BC0, BC1, and BC2 of MTA C's intracluster buffer pool may be moved to BC0 of MTA C's intercluster buffer pool and messages in BC1 of MTA C's intercluster buffer pool may be moved to BC0 of MTA C's intracluster buffer pool. MTA D is configured analogously. Buffer classes in Figure 4.2 containing a message are

[3] Refer to [Günther81] for a more detailed description of the structured buffer pool method.

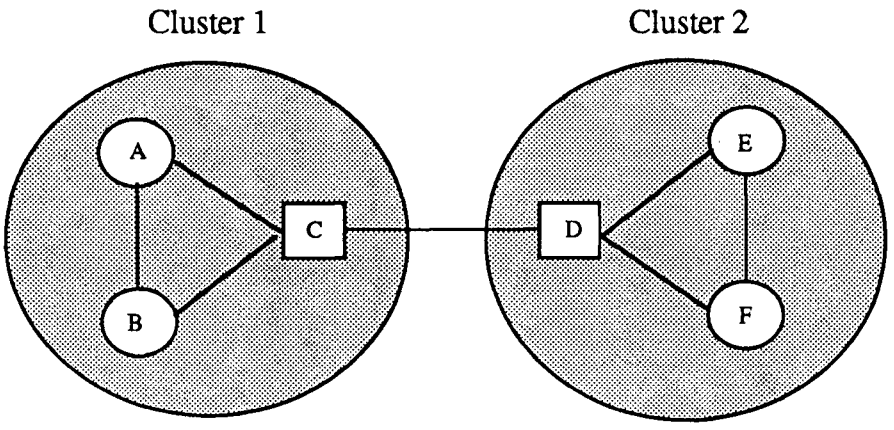

Figure 4.1: Two Interconnected Clusters

labelled with the recipient MTA of the message (*e.g.*, BC0 of MTA A contains a message addressed to MTA F) and the arrow indicates where the message is to be forwarded. The buffer pool system in Figure 4.2 is currently deadlocked with the messages in the shaded buffer classes unable to be forwarded.

An appropriate choice for the technique used within a cluster of hosts to deal with deadlock depends on the network topology of the cluster, primarily the characteristics of the network level. A cluster whose hosts are interconnected by a local area network would probably choose a different technique than one whose hosts are interconnected by a slower communication subnet. In a cluster using S/F communication at the network level, a deadlock detection and resolution technique (see [Chan87], for example) might be chosen rather than a deadlock prevention technique. The extra communication overhead incurred by deadlock detection and resolution techniques will be less significant in an appropriate

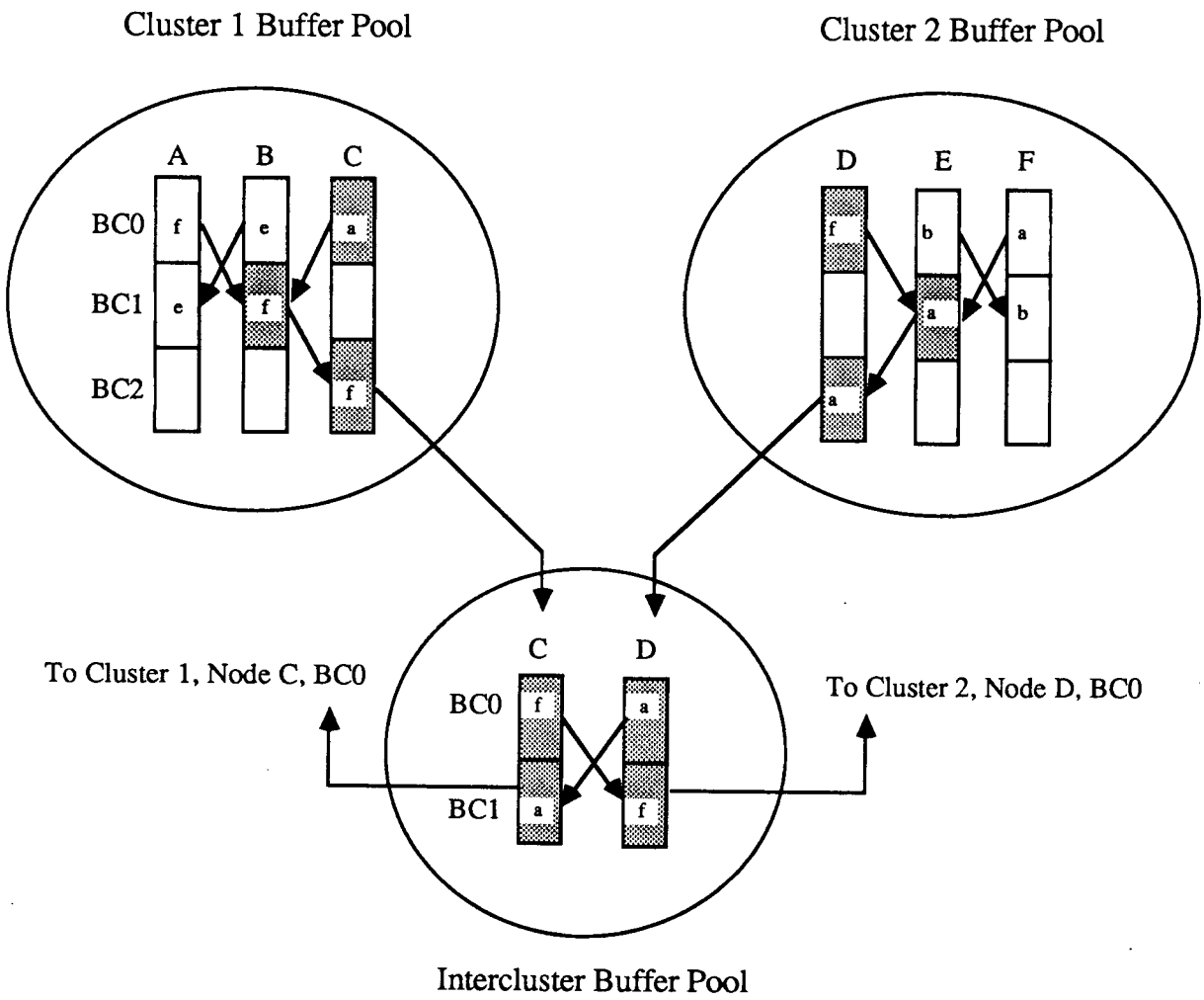

Figure 4.2: A Two-Level Structured Buffer Pool

local area environment.

In the next section the four types of intracluster message transfer are outlined. The proof that the schemes described in Section 4.2.2 are deadlock free is based on the fact that the schemes handle each of the possible message transfer types without deadlock.

4.2.1 Intracluster Message Transport

There are four types of message transport that must be handled by an intracluster message transport mechanism. This mechanism is responsible for transporting messages between any two MTAs within the same cluster. A message passing through only a cluster gateway MTA of a cluster does not use intracluster transport. The intracluster transport mechanism transfers messages where:

1. Both originator and recipient are in the same cluster.

2. Only the originator is in the cluster.

3. Only the recipient is in the cluster. A message must eventually be delivered to its recipient MTA once it has arrived at the cluster containing its recipient. If the cluster gateway is not the recipient MTA then the message must be forwarded to the recipient MTA.

4. The message is passing through a cluster; *i.e.,* neither the originator nor the recipient belong to the cluster. A message that has arrived at buffer class N at a cluster gateway and that has to be forwarded to a second cluster gateway within the same cluster for transfer to an adjacent cluster must eventually be delivered to buffer class N at the second cluster gateway. At that point the message reenters the top level system. This requirement allows the number of buffer classes to be independent of

the internal topology of clusters.

To show that a particular deadlock-free intracluster message transport mechanism, when combined with the deadlock-free top level structured buffer pool mechanism, results in a deadlock-free interconnection of systems, these cases must be handled by the intracluster mechanism while not introducing deadlock in the top level structured buffer pool system.

A feature of the hierarchical method is that a message will never be required to be routed out of a cluster to prevent deadlock; *e.g.,* messages having the originator and recipients within a single cluster will not be transferred out of the cluster to prevent deadlock. For example, if the time-stamp technique is used within a cluster, a message exchange operation will not require the message to be transferred to a different cluster to prevent deadlock. Also, messages that only need to pass through the cluster gateway do not have to be forced into any other MTAs in the cluster.

The hierarchical scheme may also be applied at lower networking levels as long as these requirements are met. For example, a packet-switched or datagram-based network may also use the hierarchical technique (see Section 4.2.5).

4.2.2 Basic Intracluster Message Transport

Two basic methods for providing intracluster message transport are discussed in this section. First, a connection-based method and then a S/F oriented method are described and both are shown to be deadlock free. These methods employ the time-stamp technique as well as a top level structured buffer pool scheme. To simplify the descriptions of the methods, the use of both fixed-length messages and buffers will be assumed. Variations of the store-and-forward oriented method that require fewer buffers are outlined in Section 4.2.3 and methods for dealing with variable-length messages are discussed in Section 4.2.4.

4.2.2.1 The Connection-Based Method

A connection-based protocol can be used in a cluster providing end-to-end communication between all pairs of MTAs within the cluster. Cluster gateways place a submitted message going to another cluster in buffer class zero of the intercluster buffer pool. When a receiving MTA within the cluster does not have any available buffer space, the time-stamp technique determines whether an exchange of messages must be performed or if the sending MTA must retry the transfer later. Each MTA has some buffers reserved for accepting messages on behalf of recipients and a message cannot occupy these buffers indefinitely. The connection-based protocol may handle the four types of message transfer as follows:

1. If both the originator and recipient are within the cluster, the originator establishes a connection to the recipient and an attempt is made to transfer the message.

2. If only the originator is within the cluster, an internal MTA establishes a connection to the appropriate cluster gateway and attempts to transfer the message to buffer class zero.

3. If only the recipient is within the cluster, the final cluster gateway attempts to transfer the message to the recipient MTA.

4. When a message arrives at a cluster gateway and must be transferred to a second cluster gateway within the same cluster, the sender establishes a connection to the receiver and attempts to transfer the message. The message retains its current buffer class at the receiving cluster gateway.

Theorem 1:

The combined top level structured buffer pool and bottom level connection-based protocol are deadlock free.

Proof:

At any instant there exists an "oldest" message M in the network. It is shown that M will be delivered to its recipient MTA within the cluster or to the appropriate buffer class of a cluster gateway where it will be transferred to the next cluster gateway.

Either M is at a cluster gateway or an internal MTA. Since M is the oldest message in the network, it can be exchanged with any other message. If M is at a cluster gateway awaiting transfer to an internal recipient MTA, it can eventually be transferred since space must become available at the recipient MTA. If M is in buffer class N at a cluster gateway awaiting transfer to buffer class N at a second cluster gateway within the cluster it can be transferred, performing an exchange with another message if necessary. If M is in buffer class N at a cluster gateway awaiting transfer out of the cluster it will eventually be transferred to buffer class $N+1$ of the next cluster gateway since it can not be exchanged with another message and because the structured buffer pool is deadlock free (*i.e.*, a buffer of buffer class $N+1$ must eventually become available at the next cluster gateway). If M is at an internal MTA, it can be transferred to a recipient internal MTA (since space must become available) or to buffer class zero of a cluster gateway, using an exchange if necessary. ■

4.2.2.2 The Store-and-Forward Oriented Method

An environment providing virtual circuits (*e.g.*, X.25) within a cluster can use the connection-based protocol. Virtual circuits that preallocate packet buffer space at each host along the route chosen at connection establishment time are deadlock free.[4] In a cluster supporting

datagrams (*e.g.,* IP datagrams), the connection-based protocol can also be used, employing a transport layer that breaks a message into datagrams, reassembles the message, and guards against transmission errors (*e.g.,* TCP/IP or a protocol specific to local area networks [Zwaenepoel85]). Datagram deadlock can be prevented by using the time-stamp technique or simply by detecting deadlock, performing a reset, and having the higher level protocol recover.

A general method of transporting messages within a cluster uses the S/F technique. The penalty paid for not being able to establish end-to-end connections is an increase in the complexity of the mechanism and increased buffer space requirements. The simplest mechanism allocates buffer space at each MTA (excepting the cluster gateways which already have this space as part of the intercluster structured buffer pool) for each buffer class (See Figure 4.3). The general S/F protocol may handle the four types of intracluster transfer as follows:

1. If both the originator and recipient are within the cluster, the message is submitted to buffer class zero and continues to use buffer class zero until it arrives at its recipient.

2. If only the originator is within the cluster, the message is submitted to buffer class zero and continues to use buffer class zero until it arrives at a cluster gateway for transmission through the intercluster mechanism.

3. If only the recipient is within the cluster, the message retains its current buffer class until it arrives at the recipient MTA.

4. A message at a cluster gateway destined for a cluster gateway within the cluster uses only buffers belonging to the buffer class it currently uses.

[4] Because the process of setting up a virtual circuit is aborted if it cannot be completed within some period of time, virtual circuit resources cannot themselves be responsible for deadlock.

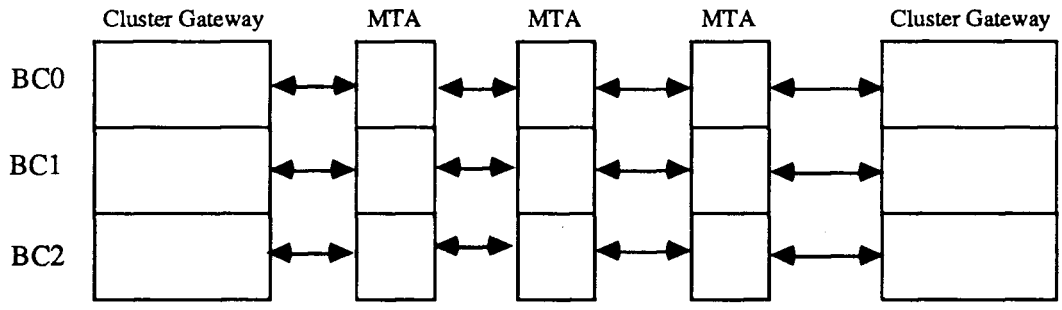

Figure 4.3: A Simple S/F Intracluster Buffer System

In a network having N MTAs and M clusters (typically $M \ll$N), the entire network would need a minimum of $N * M$ buffers instead of the $N * N$ buffers required by the pure structured buffer pool technique.

Theorem 2:

The combined top level structured buffer pool and bottom level general S/F protocol are deadlock free.

Proof:

This is only a slight variation of Proof 1, reflecting the need to store and forward a message through the cluster.

Suppose M is the oldest message in the network. If M is at a cluster gateway awaiting transfer to an internal MTA, it can be transferred in turn to each MTA on the route, retaining its current buffer class. Exchanges based on time stamps may occur. If M is in buffer class N at a cluster gateway awaiting transfer to buffer class N at a second cluster gateway within the cluster it can be transferred in turn to each MTA on the route,

performing exchanges when necessary and remaining in buffer class N at each MTA. If M is at an internal MTA, it can be transferred to a recipient internal MTA or to a cluster gateway, in either case retaining its current buffer class and using exchanges when necessary. The remaining cases are handled as in the previous proof. ∎

4.2.3 Enhancements to Basic Intracluster Message Transport

Several schemes can be used to reduce the number of buffers required by the general S/F protocol. In the common case of a cluster having a single cluster gateway, deadlock-free transport can be obtained by creating two mechanisms. The two mechanisms are needed to keep messages from being placed in an inappropriate buffer class. A *delivery transport mechanism* can be created to guarantee the eventual transfer of a message from its final cluster gateway to its recipient MTA and for transferring messages having both the originator and recipient inside the cluster. It reserves at least one buffer within each MTA of the cluster for the sole purpose of providing a "channel" for message delivery. A separate *submission transport mechanism*, requiring at least one buffer at each MTA (except the cluster gateways), transports messages to buffer class zero of a cluster gateway. The time-stamp technique can be used by either mechanism to transfer messages in a deadlock-free manner. Figure 4.4 illustrates this scheme where there is a linearly connected cluster of three MTAs and a single cluster gateway. As in the connection-based protocol, the buffer classes in the cluster gateways can be partitioned. The reduction in buffers is a result of the elimination of the requirement that the number of reserved buffers be dependent on the number of buffer classes. This is possible because messages cannot pass through the cluster.

In a cluster having multiple cluster gateways, a compromise between the two previously described S/F protocols can be reached. At least one route must be created through

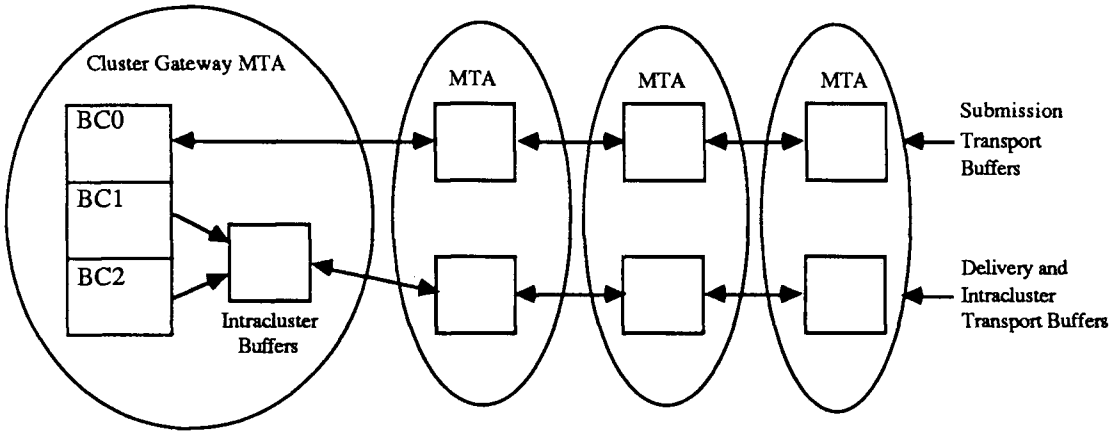

Figure 4.4: An Enhanced S/F Intracluster Buffer System

the cluster that provides a message connection for each cluster gateway to each of the other cluster gateways. Each MTA on these routes allocates buffer space for each of the buffer classes. Any MTAs not on these routes may use delivery and submission transport mechanisms (which may use less space and are independent of the number of buffer classes) to transfer to and from MTAs that have the full buffer class allocation. The amount of buffer space required may be reduced by decreasing the number of different routes connecting the cluster gateways.

4.2.4 Variable Length Messages

Although variable-length messages do not present a problem to the structured buffer pool technique [Merlin80], they are a problem for exchange-based techniques. In the time-stamp technique, buffers of equal size are required since an exchange operation may occur (*i.e.*, the two messages being exchanged must fit into each other's buffer). As an example of the problem, suppose Host X wants to send the oldest message to Host Y and Host Y does not

have enough buffer space to receive the message. According to the algorithm, an exchange operation must be performed. If the size of the oldest message plus the amount of free space at Host X is less than the size of the smallest message at Host Y then no exchange can take place. A simple but unattractive solution is to impose a network wide limit on the maximum message size and make all buffers in the message handling system this size. If large messages are to be supported this would require large amounts of buffer space that would be used inefficiently because of internal fragmentation.

Message fragmentation solves this problem since messages can be broken into equal-size fragments.

Deadlock caused by the reassembly of message fragments (*i.e.*, at the end-to-end level) must also be prevented. In cases where the time-stamp technique is impractical for this purpose, preallocation of buffer space may be used. In one scheme, the system does not allocate buffer space at the recipient MTA to any of the message until there is enough space to contain the entire reassembled message. Another scheme, appropriate for larger messages, involves reservation of space on an end-to-end basis, with the transfer being delayed until the reservation is confirmed.

Several ways to fragment messages for transport through the message handling system were discussed in Section 3.2.1. One alternative is to choose a network-wide maximum message size and fragment a message at the originator into several smaller messages. Buffers belonging to both the structured buffer pool and any exchange-based mechanisms would be of the maximum message size. This scheme can be made more elaborate by having several different message sizes, perhaps one based on the average interpersonal message size and another based on applications transferring much larger messages (*e.g.*, file transfer). Buffer space for each buffer class and exchange-based mechanisms would be partitioned into the

appropriate number of pools each consisting of buffers of the chosen size. This fits in well with a congestion control technique where preference is given to one group of traffic (*e.g.*, small messages) over another (*e.g.*, big messages) simply by controlling the allocation of buffer space.

4.2.5 Application to Packet Switched and Datagram Networks

The hierarchical method can be applied to packet-switched or datagram networks. In either environment the time-stamp technique would be used within clusters. This would, of course, incur the added expense of a time stamp header field to each packet or datagram. As in the message handling environment, the total number of buffers required would be reduced over a pure structured buffer pool method. Also, the number of exchanges and their cost would be reduced over a pure time stamp-technique.

If it is acceptable to discard packets (or datagrams), then a scheme using fewer buffer classes might be used. If desired, a higher-level protocol could be used to retransmit a discarded packet. If a packet is placed in a buffer belonging to the highest buffer class but is not at its destination, it may be put into any lower buffer class (preferably the lowest one available) provided it does not wait indefinitely for a buffer. If, after waiting the maximum allowed period, a buffer has not become available, the packet must be discarded. The resulting structured buffer pool system remains deadlock free.

The motivation for this procedure is that messages following very long paths may be fairly rare in some topologies and it is not desirable to permanently allocate buffers over the entire structured buffer pool system to handle exceptional cases (*e.g.*, rerouted or forwarded packets). Packets following a route of length less than the maximum (as determined by the number of buffer classes provided) are guaranteed to be delivered. A packet following an

unusually long route is delivered with a probability dependent on its current location and destination, and the network topology and current state. The designer must be careful that packets that follow a long route are not uniformly discarded under heavy network load.

4.3 Summary

A hierarchical approach to solving S/F deadlock has been described. By combining the advantages of the structured buffer pool approach with those of an appropriately chosen second deadlock prevention technique, a better deadlock-free transport scheme can be devised. A structured buffer pool approach is used as a top layer to transport messages between clusters of hosts while a topology-dependent technique is chosen as a bottom layer to transport messages within a cluster of hosts. The advantages of the structured buffer pool are retained while the impact of many of its negative characteristics is reduced. Although the time-stamp approach is also inappropriate as the sole means of deadlock prevention at the application level, it is well-suited as the basis of a secondary, intracluster means of deadlock prevention. The use of a structured buffer pool approach as the top layer has the additional advantage of being easily integrated with flow control and congestion control mechanisms. The hierarchical approach is useful in the messaging environment as well as in packet-switched and datagram networks.

Chapter 5

A Structure for Message Transfer Systems

A structure for the design of message transfer systems is presented in this chapter.[1] The design methodology handles arbitrary length, multirecipient messages and deals systematically with the flow and congestion control problems outlined in Section 3.5.2.

The structure takes advantage of the typical cluster topology of networks and uses a structured buffer pool organization to prevent intercluster S/F deadlock. Flow and congestion control are effected by two mechanisms:

1. The Message Stream Model, which might be described as an application level virtual circuit.

2. A buffer space allocation system used to reduce congestion and manage buffer space for incoming messages.

The task of message routing is left open and is outside the scope of this thesis. There are reasons to establish only a loose association between congestion control and message

[1] Portions of this material appear in [Brachman89a] and [Brachman89b].

routing [McQuillan79]. By placing only minor restrictions on routing, the framework allows the routing mechanism to be independently developed and tuned. One such restriction is that of the structured buffer pool on the maximum number of hops taken by a message. In general, a message stream establishes a fixed route for messages, although the means by which the route is chosen is not prescribed.

It is important to note that details of individual implementations may very well differ. Where fundamental differences in network topology must be taken into account, more than one design approach is given. Various parameters, such as fragment sizes, flow control window sizes, and the number and types of traffic groups must be tuned or optimized for individual implementations. Implementation specific aspects are left unspecified. The degree of fairness to be provided, message lifetimes, and the amount of buffer space required will vary among implementations. In many cases guidelines are given to be used in determining these constants.

The remainder of the chapter is organized as follows. Message fragmentation, message streams, congestion control, transit buffer management, and recipient buffer space allocation components of framework are discussed in Sections 5.1 through 5.5. Some of the fairness issues addressed by the framework are the subject of Section 5.6. Comments on network management appear in Section 5.7, and Section 5.8 concludes the chapter.

5.1 Message Fragmentation

Messages exceeding the maximum fragment size are broken up into fixed length fragments by the originating MTA. Each fragment is a complete message, consisting of an envelope and some portion of the original message content. No limit is imposed on the size of a submitted message by the message transfer system.

A message identifier is a unique, MHS-wide transaction identifier assigned to the message by the originating MTA. A common format for the identifier is the concatenation of the MTA name with the time and date the message was created. Message identification can be extended to uniquely identify fragments by adding a structure indicating the byte range with respect to the content of the original message; *e.g.*, (*StartingByteNumber, OriginalMessageSize*). The *OriginalMessageSize* is used by a recipient MTA to tell when it has received the last fragment of the original message as well as allowing the recipient to plan for subsequent fragments after the first arrives.

The fragment size is largely dependent on a particular network's traffic and would therefore likely vary in different implementations. A number of guidelines, however, can be observed:

- The median message size should be taken into consideration. The fragment size should be made large enough to avoid fragmentation of most messages.
- The fragment size should reflect transmission rates, particularly cluster to cluster transmission rates. Fragments that are too large relative to transmission rates will increase mean queueing delay.
- As the maximum fragment size decreases, the proportion of processing overhead per fragment increases. Increased processing demands could lead to CPU bottlenecks.
- Multiple fragment sizes (and consequently buffer sizes) may be indicated (*e.g.*, if the message size frequency distribution graph is bimodal two fragment sizes are suggested).
- As has been discussed, a maximum fragment size places a limit on the envelope size. Given a maximum envelope size, a minimum content to envelope size ratio should be

determined.

A variable length envelope size presents a problem if the system cannot fragment it and a fixed maximum fragment size is used. The number of recipients is the primary contributor to the problem. The simplest solution is to impose a limit on the length of the envelope. Some limit *must* be set in any case since otherwise the aims of message fragmentation will be defeated; *i.e.*, a message could be too large to be stored at an intermediate MTA. Since the originator will normally respond to this limitation by submitting a message as many times as necessary using subsets of the original recipient list, the message handling system should provide this functionality. Partitioning the recipient list may result in reduced efficiency since the same message may have to be sent out several times along the same route. The overhead can be reduced in some cases by partitioning recipient lists according to clusters and, if possible, using topological information. The message handling system should be in a better position to do this than the user. If the MTS appends information (such as time stamps) to the envelope, the fragment size should take this into consideration. Another approach to this problem is given in Section 5.2.1.

When all of the fragments arrive at the recipient MTA, they are reassembled to form the original message. An MTA that acts as a gateway to a message handling system that does not support message fragmentation must also perform reassembly before forwarding the entire message to the next message handling system. There is no danger of reassembly deadlock since the buffer space reservation system, discussed below, eliminates this possibility. Upon arrival of the last fragment at a recipient MTA, the recipient can be notified of the message's arrival.

Since there may be a considerable delay between the arrivals of the first and last fragments of a message, the message lifetime must be adjusted so that it is unlikely for the first

arriving fragment to expire before the arrival of the last fragment. This can be achieved by starting a timer when the first fragment arrives at the recipient MTA and resetting the timer whenever another fragment of the message arrives. If another fragment is not received before the timer exceeds some threshold (perhaps as long as the network transit time) then the transfer may be aborted. The acceptance period begins after the message has been reassembled. The choice of a timer threshold depends on factors such as the amount of available buffer space and message transfer costs.

Interconnection between a non-fragmenting MTS and a fragmenting system requires fragmentation of a message at a gateway to the fragmenting MTS. Before a fragmented message can be forwarded to a non-fragmenting system the fragments must be reassembled at a gateway MTA. Some of the implications of this with respect to buffering at the gateway are discussed later in this chapter.

Interconnection between two fragmenting systems having a different maximum fragment size also requires special attention. Fragments entering an MTS that uses a fragment size larger than the previous MTS may lead to inefficiencies and congestion resulting from internal fragmentation. If the disparity between the two fragment sizes is large, then the smaller fragments should be combined to form larger fragments. Fragments entering an MTS that uses a fragment size smaller than the previous MTS must be refragmented so that they can be transported through the MTS.

5.2 The Message Stream

A *message stream* is a mechanism for unidirectional, flow-controlled message transfer. The purpose of the message stream is to limit the number of transit buffers in concurrent use by a traffic group and to perform flow control on a per-traffic group basis. Once established,

a message stream uses a fixed route, allowing backpressure [Gerla88] to be used instead of end-to-end acknowledgements to effect flow control. Backpressure permits smooth regulation of traffic flow from originator to recipient because intermediate MTAs can gradually increase or decrease the number of buffers allocated to a message stream. The individual messages belonging to a message stream need not be delivered in sequence, although sequenced delivery could be supported; *i.e.*, message ordering could be preserved. The individual MTAs constituting a message stream need not be simultaneously operational. As in current message transfer systems, an MTA will make repeated attempts at establishing an RTS connection. Congestion control is achieved by limiting the amount of resources each message stream may hold as well as the number of active message streams.

The creation of a new traffic group in turn creates a new message stream. A message stream is known by a unique identifier that associates each message with the message stream to which it belongs. Each message stream has a set of attributes; *e.g.*, a priority attribute used in the scheduling of message transfers. A message stream may have a fragment size associated with it. Individual messages may also have associated priorities, used in scheduling transfers within a message stream. A message stream can be used for a particular transfer after the originating MTA interacts with the recipient buffer management system (described in Section 5.5).

The traffic group concept allows successive message submissions from a particular originator belonging to the same traffic group to use the same message stream. All fragments of a message normally belong to the same message stream. There can be different types of traffic groups. One type of traffic group is an individual user's submission of a message to a particular recipient. Another is an individual user's submission of a message to recipients belonging to the same cluster. In this case flow control is performed on the originator's

sequence of submissions. A third type consists of messages having the same originating cluster and going to the same recipient cluster. Consequently, flow control is performed on an originating to recipient cluster basis. An additional flow control mechanism would be required in this case to provide fair allocation of buffers belonging to the message stream.

When the submission of a message creates a new message stream, a route must be determined (several routes, in some cases) to transport the fragments of the message to each recipient. There are many alternative algorithms for selecting a route. In the case of an incrementally determined route (see [Elie79], for example), the first step is for the originating MTA to choose an outgoing cluster gateway MTA. In clusters where there is more than one cluster gateway this routing information could be distributed to the individual MTAs within the cluster or a dynamic (query-based) mechanism could be used. Each cluster gateway in turn selects the next cluster gateway until the recipient's cluster is reached. Finally, the route leads to the recipient MTA within the last cluster. Messages that must flow through a cluster (*i.e.*, entering via one cluster gateway and exiting via another) may continue to extend the same message stream through the cluster, create a new message stream handling a larger traffic group (*i.e.*, use a generic stream for carrying all "through traffic"), or use some other mechanism depending on the topology of the cluster. Each MTA along the route records the message stream identifier of the new message stream and maintains information on the state of the message stream. The algorithm used to select successive MTAs forming the route is left open. Factors such as load, buffer availability, and the number of message streams passing through an MTA could be taken into account.

An intercluster message stream uses the underlying structured buffer pool system. At each successive cluster gateway along the route the required buffer class number increases monotonically.[2] In this way the message stream is guaranteed to be deadlock free, even if

the message stream's route is incrementally generated. Furthermore, it is not necessary to dedicate any buffers to the message stream. Without an underlying structured buffer pool, determining a route in this way is more complicated and less efficient if deadlock is to be avoided.

There are several topology-dependent options available for intracluster message transport. Like the intercluster message stream, an intracluster message stream can be established if there is an underlying structured buffer pool. A structured buffer pool may be appropriate if the maximum number of hops to traverse the cluster is small. In general, for a network consisting of C clusters and a cluster having N MTAs, the cluster will require $C * N^2$ buffer classes for a complete structured buffer pool system. Each MTA's structured buffer pool is hierarchically structured, with one "top"level buffer class for each of the C clusters and N "bottom" level buffer classes within each of the top level buffer classes. The top level buffer class used by a particular message depends on the number of clusters the message has traversed and the bottom level buffer class depends on the number of MTAs the message has traversed within the current cluster.

In some clusters the number of buffer classes required might be too large and so another method is required. Unfortunately, when the time-stamp exchange technique is used within a cluster there does not appear to be a simple way to construct a message stream that maintains proper backpressure. This is because a time-stamp exchange may move messages off the message stream's route to prevent deadlock, destroying the flow control aspect of the message stream. While it is possible to limit the use of the time-stamp exchange to route selection, hop-by-hop allocation of buffers is not deadlock free. An alternative is to establish the message stream by simulating a virtual circuit "call setup" operation to

[2] The simple hop count technique is assumed here to select buffer classes: a new message is placed in buffer class zero and a message currently using buffer class N requires buffer class $N + 1$ at the next MTA.

select a route and reserve buffers at each MTA on the route. If space cannot be obtained at an MTA on the route, another route can be considered. If a message stream cannot be established then all allocated buffers must be released and the call setup aborted.

With this scheme, a large cluster using S/F message transfer will have difficulty establishing a message stream (especially a multirecipient message stream involving many MTAs) under heavy load. Small clusters, clusters not under heavy load, and clusters having connection-based facilities should function well, however.

Another alternative is to use a simple acknowledgement scheme where the sender pauses after N messages have been sent, where N is cluster dependent. The sender continues after receiving an acknowledgement. There can be a substantial amount of bookkeeping associated with multirecipient messages directed to many MTAs since the sender must manage the flow control of each receiving MTA. There can also be tradeoffs between transport efficiency and delay.

The lifetime of a message stream can be relative to the number of messages transferred, relative to the time and date the message stream was established at each MTA along the route, or relative to the time and date of the last forwarded message belonging to the message stream (*e.g.,* the length of time the message stream has been idle). Once the lifetime of a message stream has elapsed, no more messages can be sent via the message stream. A message stream may be shut down in an incremental fashion by sending a control message. Any reserved resources are deallocated when the message stream is shut down. Message streams are given a limited lifetime so that any reserved resources can be put to other uses and to allow later instances of the traffic group to use a message stream following a different, and possibly better route. When message streams are allocated transit buffers for their exclusive use, it may be wise to choose shorter lifetimes. The lifetimes

may be assigned on a per message stream basis and the method of determining lifetimes for intracluster message streams need not use the same criteria as intercluster message streams.

The message stream for a multirecipient message is formed by splitting the stream at various points along the route, creating one or more branches that may in turn be split.[3] An MTA that splits a message stream keeps track of the continuation of the message stream (*i.e.*, the set of adjacent MTAs to which the message must be forwarded). A consequence of managing transit buffer space strictly on a per message stream basis is that if the flow of any branch of the message stream is stopped all upstream traffic will also eventually be stopped, even if flow down other branches is possible. This can be ameliorated, for example, by allocating additional transit buffers to the message stream at these MTAs so that copies of the fragments that could be forwarded down unblocked branches can be retained until the flow is resumed down the other branches. Obviously the amount of this allocation must be carefully considered so that transit buffers are not held for too long. Also, if the obstruction lasts beyond the lifetime of the branch of the message stream responsible for the congestion, the branch will be shut down.

A disadvantage of restricting transmission of all fragments of a message to the same route is the inability to increase parallelism by routing each fragment independently, as is done in datagram networks. Increased parallelism in message stream transfers can be achieved, however, by establishing multiple message streams. Individual streams need not follow the same route.[4]The message stream identifier can indicate a particular multiple message stream transfer as well as the total number of multiple message streams involved.

[3] As a special case, a branch of a message stream may be of length zero; *i.e.*, messages may be "dropped off" at a recipient MTA along the route of the message stream.

[4] Multiple message streams may, however, be constrained to terminate at the same gateway MTA (Section 5.5).

In the event a message stream's flow is halted because of a failure, it may be possible to create an additional message stream to bypass the failure. A more complicated alternative is to allow a message stream to be rerouted around failures. Rerouting can be accomplished by splicing in a new path to bypass the inaccessible part of the route. Any messages an offline MTA might be holding would be forwarded when the MTA becomes available. The same mechanisms used in the case of failures may be used to bypass areas of congestion.

A switching technique similar to virtual cut-through [Kermani79] can be used to reduce total transfer delay and transit buffer requirements. In some cases it may be possible to begin forwarding a message to the next MTA on the route after only the envelope of the message has been received. To provide reliability or increased efficiency, a copy of each message fragment must still be retained at the sending MTA until the fragment is successfully forwarded. Also, it may be the case that not all of the set of receiving MTAs on a multirecipient message's route are able to receive at the same time and so the message must be buffered. An MTA can transparently choose to use this technique on any outgoing link identified as being suitable. For topologies supporting rapid connection establishment, the technique could prove useful for real-time transfers.

5.2.1 Envelope Fragmentation

It was previously noted that a variable length envelope may defeat the goals of message fragmentation. It was argued that since there must be a limit on the envelope size, a reasonable solution is simply to set the fragment size to reflect this. It has been assumed that each fragment consists of an envelope virtually identical to that of the submitted message and some portion of the submitted message's content. If the constraint on the format of a message fragment is relaxed, an arbitrarily large envelope can be supported by a fragmenting message transfer system. Following X.400, a fourth type of information

object can be introduced to formally describe the message fragment.[5]

Envelope fragmentation can be achieved by associating the submitted message's envelope with the message stream's state information. The submitted message's envelope is fragmented and sent down the message stream, preceding the message's content. While the ordering of the fragments of the message content may be permuted, the envelope must continue to precede the content. A fragment containing content from the submitted message has a short, fixed length envelope, identifying the message stream to which it belongs. This approach is more efficient since the envelope is sent once instead of as part of every fragment.

To avoid deadlock, before a receiving MTA can accept any portion of the fragmented envelope during the message stream construction phase it must be able to store the entire reassembled envelope. This is also necessary for message routing purposes. This is the only constraint on the length of the envelope. An implementation would likely establish a MHS-wide maximum envelope size for reasons of fairness.

5.2.2 Message Stream Multiplexing

An MTA may establish many simultaneously active RTS connections[6] if topology permits. Because there is usually a limit on the number of active RTS connections and some network interfaces are shared (*e.g.*, modems), message streams may have to take turns establishing RTS connections. Message streams containing messages to be forwarded to the same adjacent MTA can be multiplexed onto the same RTS connection. Message priorities are used to determine the scheduling of message transfers. It is possible, but not necessary, to assign priorities to message streams so that message scheduling is within message stream

[5] The other types, described in Section 2.1.1, are the message, probe, and report.

[6] RTS connections were described in Section 2.1.

scheduling.

The scheduling of message transfers can become complicated and difficult to implement given a network using many types of communication devices providing different degrees of sharing. Message transfers can be queued taking into account such things as message priority, message stream type, receiving MTA, and RTS connection characteristics.

The tasks of determining the next RTS connection to establish and the next message to forward on an RTS connection are performed by the message scheduler. Message transfers are multiplexed using a prioritized round-robin scheduling algorithm. Multiplexing is performed first on message streams and then on messages within streams. The task of the scheduling algorithm is to provide fair (as determined by priorities) and livelock-free message forwarding. Message scheduling can be preemptive but would seem unnecessary if the maximum fragment size is not large relative to the transmission speed.

Each message stream and individual message is assigned a priority P, $1 \leq P \leq N$. A message with priority P is placed in the corresponding queue in an N level queueing system. Ready transfers are multiplexed so that a transfer with priority P will be scheduled approximately P times more frequently than ready transfers with priority 1. Round-robin scheduling is used within a queue level. A transfer that loses its turn to an arriving higher priority transfer has its priority increased so that eventually all transfers will be scheduled. Transfers assigned a high priority can, of course, result in long waiting times for transfers assigned a low priority.

Another factor to be considered involves a tradeoff between cost and delay. If there is a charge associated with establishing an RTS connection (*e.g.*, there is often a minimum charge for a long distance telephone call) then in some cases it can be economical to wait until the queue length grows before the RTS connection is established [King86]. On the

other hand, delaying transfers to take advantage of lower toll costs may require additional input and transit buffers.

5.3 Congestion Control

Congestion control is effected by limiting the number of message streams in existence at the originating MTA. This limit can be based on periodic determination of the input buffer limit. Further restrictions can be placed on the number of message streams belonging to a particular originator, leaving a cluster, or going to a particular recipient MTA or cluster; *e.g.,* the amount of input buffer space made available to a particular traffic group can be proportional to the traffic group's throughput. Again, the particular restrictions imposed determine the tradeoff between fairness and sharing.

Requests for new message streams fail if the network is deemed to be congested, if one of the limits on the number of message streams of that type has been reached, or simply if there is no input buffer space. The number of transit buffers available to input traffic is adjusted according to traffic conditions, effectively limiting the number of message streams.

The structured buffer pool system, with the dynamic allocation of transit buffers to buffer classes, and recipient buffer space management (Section 5.5) play a significant role in reducing congestion. This is demonstrated in the results of the simulation study discussed in the next chapter. The message stream mechanism also tends to reduce congestion because the transit buffer usage of a message stream is limited at each intermediate MTA.

Increased global knowledge can be used to provide better congestion control. This might include, for example, information concerning details of the interconnection of cluster gateways. Periodic distribution of message stream statistics and buffer usage can assist in determining bottlenecks in the network.

5.4 Buffer Management

If the most important uses of the MHS are seen to involve the transfer of smaller messages (*e.g.*, interpersonal messaging, queries, *etc.*) then steps must be taken in the design of the MHS to see that large transfers do not unfairly impede smaller ones. Large transfers have much higher demands for buffer space and connection time. Of primary importance is that the network should avoid allocating significant quantities of transit buffer space to a large transfer. Apart from fairness considerations, a stalled transfer may tie up transit buffer space that could better be used by more "successful" transfers.

One solution to buffer space contention problems between large and small transfers is to provide separate buffer pools for the two classes of traffic. To avoid the situation where one pool has free space while the other is full, each buffer pool could be given a minimum allocation and a third, shared buffer pool could be established. The number of buffers available to each pool could be dynamically adjusted based on traffic conditions.

Buffer space must be allocated to each MTA for buffering input, transit, and received messages. The management of buffer space for received messages is discussed in Section 5.5. Management of submitted messages plays an important role in offline transfer since a message may need to be stored for some time before it can be forwarded. The usual method is to place a copy of a message submitted to the message transfer system in a shared buffer pool. Once the input buffer pool is filled, the system cannot accept additional submissions until some of the messages are forwarded. In some environments a quota system may be required to insure equitable sharing of the input buffer area.

As a UA prepares a message for submission to an MTA, it often copies one or more files into the body of the message. For applications such as file transfer, making a copy of

parts of the message content is wasteful of input buffer space. For example, the EANft UA builds a file transfer message by copying each file into the body of the message. Duplication can be avoided by allowing an MTA to access a file at the time of transmission (*i.e.*, when an RTS connection is established). Some assurance is required, however, that the files will not be modified between when the message is submitted and when the MTA is no longer responsible for its delivery. Operating systems that do not support file locking may not be able to prevent modification of these "included" files, necessitating copying by the UA. One solution on these systems is to let the user specify whether a copy should be made.

Given that it is safe to avoid copying a file, a UA can submit both the message and information on how to insert files into the message. At transmission time, the MTA switches input streams as necessary from the submitted message to any referenced files. This scheme is application independent since the MTA does not need to know the message format. The UA could also specify a process to be invoked to do format conversions at transmission time (*e.g.*, compression or encryption).

The message transfer system is assumed to be "reliable enough" that the destruction of a message or message fragment through a catastrophe is unlikely. A message may therefore be deleted by the originating MTA immediately after it has been forwarded. Retaining a copy of a message at the originating MTA until delivery has been confirmed allows the system to retransmit a message that is destroyed by a catastrophe. This additional reliability increases input buffer requirements. If the increased degree of reliability is not deemed necessary for all messages, it could be requested when the message is submitted. In the event a catastrophe occurs when the originating MTA does not hold a copy of the message, the message transfer system could notify the originator (suppressing multiple notifications) and discard any remaining fragments. Any remaining fragments can be

discarded by the expiry mechanism and no special action need be taken unless quicker purging is desired. Duplicate messages that arrive during the lifetime of a message stream can easily be detected. Duplicates that arrive after a message stream has been shut down can be detected if a recent history of message stream identifiers is retained.

The amount of space given to transit buffers depends on both global and local considerations. Global considerations include the maximum number of hops allowed for any message (for the structured buffer pool), the number of different fragment sizes, and the degree of fairness to be provided. A particular MTA must also be configured taking into account the amount of message traffic anticipated and the function of the MTA (a cluster gateway and a gateway MTA will need more space than a leaf MTA).

The intercluster buffer pool is broken up into partitions. At least one buffer of each fragment size must be allocated to each buffer class. Some minimum amount of each buffer class might also be dedicated to high priority message streams and control messages. The remaining space makes up a free pool that may be dynamically assigned to incoming messages.

The number of transit buffers assigned to each buffer class and individual message stream is adjusted (within MTA specific minimum and maximum limits) according to the message flow through the buffer class or message stream. As has been discussed, a non-zero minimum allocation to each message stream increases fairness while simultaneously reducing transit delay. Dedication of buffers to a message stream may reduce the degree of sharing, however.

One approach to determining the allocation of transit buffers to buffer classes is to assign a limit to the size of a buffer class based on the ratio of the throughput of the buffer class to the total throughput of the MTA. The goal of this approach is to maintain high

throughput under changing traffic conditions by periodically recomputing this ratio. Many of the details of this approach, described below, differ from those proposed previously (*e.g.,* [Giessler81]). In particular, the goal of this approach is to improve throughput. Buffer class limits are determined based on the relative throughputs of the buffer classes and are adjusted differently.

If the number of buffers in buffer class X is denoted as B_X, the total number of buffers as B_T, the throughput of messages in buffer class X as σ_X, the total throughput as σ_T, and the number of buffer classes as N (labelled $0, \ldots, N-1$), the fraction of the total number of buffers available to buffer class X is bounded by:

$$\frac{B_X}{B_T} < \frac{\sigma_X}{\sigma_T} \tag{5.1}$$

By keeping track of the throughput of each buffer class and the total message throughput over an appropriate interval, an MTA could use expression 5.1 to adjust the size of each buffer class over the next interval.

Each buffer class has variables associated with it indicating the buffer class limit, current usage, and a delta value. After some number of buffers have been released, expression (5.1) is used to compute a delta value for each buffer class indicating how much the limit should be increased or decreased. Buffer classes are initially assigned a limit such that the limits sum to the total number of buffers available. A buffer class limit can only be increased if another is decreased and a limit cannot be decreased to less than either the current usage or a non-zero minimum value.

After the delta values are updated and also after each buffer release, the vector of delta values is scanned to determine if there is both a limit that can be decreased and a limit that can be increased. A limit is eligible for being increased if its delta value is positive, its usage has reached its limit, and messages could not be accepted from an adjacent MTA

during the last interval because the limit had been reached. This scan is from the highest buffer class to the lowest since in most topologies the highest buffer classes are used less frequently and should therefore be reduced first. Also, increasing the limits of higher buffer classes first will tend to remove messages from the network faster since messages through them are closer to their recipient MTAs. The scanning is repeated until no limit can be changed.

Since neither the message arrival rate nor the throughput rate is typically a constant, this technique is likely to be sensitive to the length of the measurement interval between updates. An interval that is too short may cause too much overhead in the mechanism while an interval that is too long may not adjust to changing conditions fast enough to be of benefit. Therefore, a variable length interval that is a function of the message throughput is used rather than a fixed time interval. Giessler *et al.* [Giessler81] used a mechanism based on counters that determines the number of buffer release events required before the size of a buffer class can be adjusted. Some memory of throughput during recent intervals may be retained to smooth out changes in the limits.

After determining the adjustments that must be made to the individual buffer classes, message streams within each buffer class are examined to determine how the buffer allocation to individual message streams must be changed to reflect these adjustments. For each buffer class, expression 5.1 may be applied to the buffer class throughput and message streams using that buffer class (a message stream can only be using a single buffer class at any MTA). In this way the proportion of the buffer class that should be allocated to individual message streams can be determined. Minimum and maximum message stream allocations must be imposed.

5.5 Recipient Buffer Space Allocation

The potentially long lifetime of messages within the network, coupled with the prohibition on discarding messages, implies that the commitment of transit buffers to a transfer must be carefully controlled. The scenario to be avoided is that of a transfer unable to proceed because there is not enough buffer space (*recipient buffer space*) for the entire message at a recipient's MTA. When message traffic is heavy or when large transfers are involved such stalled transfers could tie up many transit buffers for a considerable time.

In the simplest case, the arrival of a message causes an MTA to invoke a recipient-dependent UA process for each local recipient, passing a copy of the message to each UA. After all UAs have accepted a copy of the message, the MTA deletes its copy. If the message is not claimed within the acceptance period (*e.g.*, because a UA is not available or the UA has no space to store the message) it is discarded.

This simplistic approach has two important drawbacks. The first is that a message ties up recipient buffer space while it waits to be accepted. In the worst case, the buffer space is held for the duration of the acceptance period. Without some means of prevention, it is possible for some users to monopolize recipient buffer space if their messages are not accepted in a timely manner. The second problem with this simple approach, at least in some environments, is the generation of multiple copies of a multirecipient message. A more efficient scheme would pass only a pointer or capability to the message to the UA and have the MTA act as a message store.

Given a need to manage recipient buffer space for efficiency and fairness reasons, major aspects to be considered in the design of a recipient buffer space reservation system include:

1. the delay in acquiring recipient buffer space,

2. fairness,

3. utilization of recipient buffer space, and

4. overhead of the system.

An ideal system would inexpensively grant recipients a "fair" proportion of the available space (*e.g.*, by imposing quotas) while minimizing the delay incurred by transfers and maximizing buffer space utilization. Unfortunately, given wide variations in end-to-end transit times and a nonstatic population of network users, this ideal is difficult to achieve. As might be expected, there is a tradeoff between fairness and utilization: increased fairness in sharing recipient buffer space among each of an MTA's users implies lower utilization of the buffer space. Likewise, there is a tradeoff between fairness and delay. Only a recipient's MTA can know the buffer usage of the recipient at any instant, implying that fair usage of recipient buffer space requires interaction with a recipient's MTA before a message can be submitted for transfer.

Alternative solutions to the problem of recipient buffer space allocation are discussed in the next three sections. A recipient buffer space management system is described in Section 5.5.4.

5.5.1 Preallocation

One approach is for the originating MTA to send a control message indicating the size of the transfer to each recipient's MTA. The originating MTA waits for a response from each recipient's MTA indicating that the transfer can proceed. A recipient MTA responds immediately if the message is rejected or if space is available, otherwise the request is queued and a response is sent after the MTA has allocated space to hold the message. Note

that this is not the same as a probe, the result of which indicates whether the described message could at some point in time be transferred and delivered to a recipient if it were to be submitted but does not guarantee subsequent delivery.

Potential consequences of preallocating recipient buffer space include:

1. reduced congestion since transfers do not stall because a recipient cannot accept a message due to insufficient buffer space, fairness constraints, or prevention of reassembly deadlock.

2. reduced congestion and expense since a transfer can be "refused" before it begins by the intended recipient.

3. recipient buffer space can be allocated to users according to fairness criteria.

4. more complicated, less efficient, and slower transfers than the case of no preallocation because control messages must be exchanged between the originating MTA and the recipient MTAs.

5. poorer utilization of recipient buffer space since buffers remain idle while they are reserved.

Preallocation of recipient buffer space for large transfers is desirable in that it can decrease the time to deliver the message, thus reducing the likelihood of large amounts of transit buffer space being tied up until the recipient can accept the entire message. Preallocation also solves the problem of reassembly deadlock.

Another advantage of preallocation occurs in its relationship with flow control in multirecipient message transfer. Multirecipient messages can create a conflict between flow control and efficiency considerations, as illustrated by the following example. Consider the

case of a large message addressed to two recipients within the same cluster. It is most time and cost efficient to send a single copy of the message to the destination cluster. It is also space efficient since there need only be a single copy of the message. These efficiency considerations can, however, defeat flow control of an individual's messages since recipient buffer space may not be available for one of the recipients. After being delivered to one recipient the message must be retained in transit buffers until the other recipient can accept the message.

The preallocation system can easily be extended to support a weak form of "atomic message transfer" where a transfer to multiple recipients can be aborted if any of the potential recipients are unable to receive the message; *i.e.,* if any potential recipient can not or will not receive the message then the message is not sent to any recipient. Here, the request message also serves as a probe and if any request is undeliverable (*e.g.,* an unknown O/R name was specified) or if a reply to a request is negative, the entire transfer can be aborted at the originator's option. Although "stronger" than a system based on probes, there is no guarantee that all recipients will subsequently receive the message unless the MTS is reliable.

Unfortunately, a preallocation system can lead to inefficiencies when there is more than one recipient. The transfer may not proceed optimally (*i.e.,* with the fewest possible connections and bytes transferred) unless the originator waits for buffer space for each of the recipients. There are also delays experienced by the transfer while the message requesting buffer space is delivered to each recipient MTA and the reply is returned. Another inefficiency is that this scheme may tie up large amounts of buffer space at the recipient host while it waits for the message to arrive. If the transfer is aborted the reservation of space must be released, by use of an expiry date on the reservation or an explicit control

message, for example.

Preallocation makes it easy for a user to refuse certain messages before they are ever sent. A user can specify that a request for recipient buffer space should be denied if it comes from a particular originator, if it arrives during some time period (*e.g.,* when the user is on holidays), or based on the size of the request, for example. This saves resources that would otherwise be used in sending an unwanted message and reduces message transfer system overhead.

When a recipient MTA lies outside of the message transfer system, buffer space must be reserved at a gateway MTA for message reassembly rather than at the recipient MTA. The originating MTA may explicitly select a gateway MTA or a gateway MTA may intercept the control message.

5.5.2 Optimistic Transfer

In contrast to preallocation of recipient buffer space, an optimistic approach assumes that most of the time recipient buffer space will be available when the message arrives at a recipient MTA. A message is submitted for transfer and a copy is retained at the originating MTA. A recipient MTA replies to the originator, indicating that the transfer was successful or that the message should be sent again since there was no recipient buffer space available the first time and the message had to be discarded. If a recipient MTA lies outside of the message transfer system, the gateway MTA that received the message is responsible for this buffer management.

This optimistic approach tends to reduce transfer delay but increase both cost and traffic volume. It also has increased input buffer requirements over other approaches since a copy of a message must be retained at the originator in case it must be retransmitted.

For these reasons the technique is probably not well suited to large transfers. In particular, optimistic transfer may be inappropriate for fragmented messages because if fragments are not discarded transit buffers may remain occupied for an extended period.

A hybrid, adaptive scheme may offer better overall performance than either of the preallocation or optimistic approaches. Optimistic transfer is selected initially for non-fragmented messages up to a certain size. By keeping a record of the outcome of an optimistic transfer between originator/recipient pairs over a recent time interval, the probability of success can be estimated. A probabilistic algorithm can then be used to decide whether the optimistic or preallocation technique should be used. Transfers that tend to enjoy success with optimistic approach will continue to use it or else tend to the preallocation technique.

5.5.3 Multirecipient Messages Having a Common Recipient MTA

When there is only one recipient at a particular MTA, quota management is simple since the message only has one "owner" responsible for the recipient buffer space used by the message. Assessing responsibility is more difficult in the case of a multirecipient message. How is this multirecipient buffer space accounted for and to what degree can fairness be exchanged for efficiency?

Tighter adherence to fairness is achieved through the use of "hard" quotas rather than "soft" quotas. Whereas a hard quota system never allows a user to exceed resource limits, a soft quota system allows limits to be temporarily exceeded.

At one extreme, each of N recipients of a particular M byte message might require enough of an allocation to store the message, giving each recipient a copy when the message arrives. The transfer is refused if any of the N local recipients does not have enough of

a quota reserve. This results in wasted recipient buffer space (in a global sense) and may increase delays (discussed in Section 5.5.1), but only requires a simple hard quota system.

At the other extreme, a message might be accepted if there is enough space in a common buffer pool. With no quota system, some groups of recipients could monopolize the common buffer pool. Buffer space is more effectively utilized, however, and delays are reduced over the previous method.

A scheme falling between these two extremes involves accepting a message if there is a total of M bytes available among the N recipients. This is space efficient and has good delay characteristics but results in a relaxed definition of fairness; *e.g.*, what is done if some of the recipients have already reached their quota when the message arrives and how are the individual recipient's usages recalculated when one recipient logically deletes the message? Either soft or hard quotas may be used in this scheme. If hard quotas are used, a degree of fairness is given up since space usage cannot be recalculated until all recipients are finished with the message (*i.e.*, logical message deletion may not decrease a recipient's usage).

5.5.4 A Recipient Buffer Management System

This section describes a recipient buffer management system based on the ideas presented in the previous sections. The functionality of this mechanism is a superset of that provided by the X.400/1988 message store. The primary function of the message store is to accept delivery of messages on behalf of a single recipient and to retain them for subsequent retrieval by the recipient's UA [CCITT88c].

Each user known to the message transfer system is allocated buffer space to hold incoming messages. Some local convention must be adopted to manage the use of this space

among the message transfer system and various UAs that accept and delete messages. Depending on the implementation, a user's personal allocation may be owned (with respect to the host's operating system and quota system) by the user or may be part of a common area (the *shared recipient buffer pool*) with a hard limit imposed by the message transfer system itself. This per-user buffer space is used solely by messages having no other recipients at the MTA. The respective allocations may meet any local fairness criteria, but must take into account the volume of messages expected, the largest message that can be received by a user, and the acceptance period. Messages that would raise a recipient's usage above quota are refused; *i.e.*, requests are queued and messages not using the preallocation system are treated as undeliverable.

Multirecipient messages are stored in the shared recipient buffer pool managed by the message transfer system. Users are assigned soft limits by the local system administrator. If there is enough space available in the pool and the sum of the space available to each recipient equals or exceeds the message size, the message may be accepted. An equal proportion of the size of the message is added to each recipient's usage. When a recipient is no longer considered to be responsible for a message, the portion of the total message size previously deducted from the recipient's usage is restored. When only a single recipient is responsible for the message, the space could be debited from the recipient's personal allocation, if possible. If there are no remaining responsible recipients, the message may be deleted; otherwise each recipient's usage is recalculated, taking into account the increased responsibility for the message. When usage exceeds the soft limit, action may be taken as warranted.

The hybrid technique discussed in Section 5.5.2 is used for messages considered to be "small" (a network administration decision), while preallocation is used for larger messages.

The choice of an appropriate "small" message size would likely mesh with the message transfer system's choice of transit buffer sizes. Control messages use optimistic transfer.

In the case of intercluster transfers, the delay incurred by preallocation can be reduced in exchange for allowing some transfers to stall. For example, a transfer might be allowed to proceed without a preallocation until it reaches its outgoing cluster gateway MTA. The cluster gateway then arranges the preallocation before the transfer may continue. For this approach to be useful the number of stalled transfers must be controlled and the number of transit buffers allocated to the message streams involved must be limited. Delay can be reduced in some cases by having MTAs notify their cluster gateway when recipients have reached their quota or are refusing messages. An MTA that will be unavailable for an extended period may notify its cluster gateway.

Another method of reducing delay involves making the shared recipient pool (or some portion of it) available for general use. Credits for recipient buffer space are distributed to reduce the average delay incurred in acquiring recipient buffer space and to lower overhead. Possession of credit for recipient buffer space at a particular recipient MTA insures that there is enough space at the recipient MTA before a transfer begins. A credit system may use soft quotas to manage the shared recipient pool. This approach will be most effective when there is a significant amount of message traffic between the two MTAs involved; *e.g.*, in servicing distribution lists (mailing lists). Credits are not earmarked for any special use and can be used by any message stream.

Credits are obtained in response to sending a credit request message (a type of control message) to the recipient MTA. The request includes the amount of space required and a reply confirms the allocation. To avoid reassembly deadlock, a transfer cannot be started until there is enough credit at the recipient MTA to hold the entire message. Credits are

cached to reduce the amount of credit-related message traffic. The amount of recipient buffer space requested or granted may be larger than is necessary at the time the request is made. A lifetime can be imposed on credits to increase buffer utilization.

5.6 Fairness Issues

Several important fairness issues are explicitly addressed by the resource management components described in this chapter. These elements of fairness are not present in current message transfer systems.

Foremost among these is the use of fragmentation and round-robin scheduling to significantly reduce the mean queueing delay experienced by smaller messages. Queueing delay becomes dependent on the fragment size instead of the original message size. Two severe forms of unfairness are handled; deadlock is prevented by the hierarchical deadlock prevention method and livelock is eliminated by the message stream multiplexing algorithm.

Wong *et al.* [Wong82] have developed measures of fairness in terms of relative end-to-end delays experienced by different classes of traffic in packet-switching networks. These results can be applied to the message scheduling algorithm should this form of fairness be important in the MTS.

Another significant element of fairness comes as a result of message stream based flow and congestion control. The number of transit buffers in use by a traffic group at an intermediate MTA is strictly regulated by a message stream, preventing monopolization (*e.g.*, as illustrated by Figure 3.3). Flow control is performed on a per message stream basis so that individual traffic groups are throttled back. Control can also be exercised by an originating MTA over the number of active message streams belonging to a particular originator (and also the amount of input buffer space being used if it must be shared) and

by a cluster gateway MTA over the number of message streams belonging to a particular originator that leave the cluster.

The recipient buffer management system helps to reduce unfairness resulting from excessive buffer occupancy. Recipient buffer space is managed more fairly, controlling the hogging of buffers used to store received messages.

As has been mentioned in Chapter 3 and earlier in this chapter, there are often tradeoffs between fairness, sharing, and delay. A message transfer system could allow some user control over the level of performance given to a transfer in exchange for an adjustment in the cost of the service. For example, a high priority transfer might be allowed to use more transit buffers and receive a higher scheduling priority than other transfers at some additional expense.

5.7 Tuning, Maintenance, and Disaster Recovery

The need to perform day-to-day network management, as well as changing network topology and message traffic characteristics require that provisions be made to collect performance statistics, monitor the state of the network, adjust parameters controlling MTAs, access accounting data, and so on. These management functions are needed both by personnel managing clusters and those managing the entire message handling system. Control messages can be used to perform many functions remotely, including requests for data to be sent by a return message. There is growing interest in network management systems and protocols [Feridun88, Partridge88, Case88] and standardization efforts are underway [Klerer88].

While subsystems of a message transfer system may be designed to take care of crucial problems such as deadlock prevention, there is no guarantee that a particular implemen-

tation follows the design correctly, that there are no bugs in any of the tools used in the implementation, or that hardware problems won't arise. An implementation must allow for manual intervention to correct any unusual situations. Control messages play an important role here too, allowing a central authority to diagnose and solve some types of problems remotely.

5.8 Summary

Performance aspects of message transfer systems have received little attention even though these systems have existed for some time and are coming into widespread use. In particular, flow control and congestion control present many problems. Current systems tend to ignore many of the difficult aspects of message transfer discussed here. A framework for designing message transfer systems based on message streams has been presented that addresses these problems in a structured way.

Chapter 6

Simulation Experiments

This chapter examines performance aspects of the message stream mechanism together with recipient buffer space reservations in comparison to a method of message transfer that uses neither technique. A simulation study was undertaken to evaluate some of the proposals of Chapter 5 in several network topologies while varying configuration parameters. The sensitivity of the performance to workload parameters was also studied.

One objective of the performance study was to determine end-to-end throughput characteristics of message streams in comparison to a simple, non-fragmenting scheme that does not use buffer reservations. Other objectives included gaining a better understanding of buffer class management, message stream operation, and implementation issues.

Because of the complexity resulting from the many parameters and distribution-driven components of the model, discrete-event simulation was chosen over an analytic modeling approach. A discrete-event model is flexible, easy to modify, and can capture much of the detail of a complicated system. An equivalent analytic model would likely be intractable. The ability to inspect the state of a simulation in progress proved valuable in locating bottlenecks and erroneous assumptions that might otherwise have been overlooked.

The simulation system is described in the next section. Section 6.2 discusses the methodology used in the experiments and the results are presented in Section 6.3. Conclusions of the study are summarized in Section 6.4.

6.1 The Xsim Discrete-Event Simulator

Xsim [Brachman88b] is a general purpose discrete-event simulator based on elements of the Xinu operating system [Comer84] and on the extended programming language approach to simulation, GASP II [Pritsker69] in particular. Xsim runs under Unix 4.2 BSD, 4.3 BSD, and their derivatives on the Digital Equipment Corporation VAX or Sun Microsystems Sun 3 architectures. It consists of a library of functions to create and manage lightweight processes and queues, record statistics, and generate variates from several different distributions. The simulation functions were initially tested by running several problems with known analytical solutions. Before its use in this study, the simulator had been used for several smaller projects, including another network simulation.

Xinu (version 6) [Comer84] is a process-oriented operating system that runs on PDP-11 and LSI-11 minicomputers. It provides process management and coordination, message passing, memory management, a disk driver and file system, and a store-and-forward ring network of Xinu machines. While Xsim borrows elements of Xinu that deal with process management, its processes use Unix I/O and memory allocation functions rather than their Xinu counterparts.

Xsim consists of about 50 lines of assembler code, 2000 lines of C code that perform simulation functions, and another 2100 lines of modified Xinu C code. Xsim processes run within a single Unix address space.

A combination of approaches to discrete-event simulation are used in Xsim [Fishman73,

Gordon78, Ferrari78]. In the *event-scheduling* approach, the *next event* technique is used to advance the simulation clock by a variable number of steps, skipping intervals containing no events. In the *activity-scanning* approach, events may be scheduled to take place when specified conditions are satisfied. Xsim supports both techniques (resulting in a *variable-step, activity-scanning* approach) by allowing events to be scheduled implicitly using queues or explicitly by waiting for the simulation clock to be incremented by a given amount or until an arbitrary condition is satisfied.

Queues are an interprocess communication mechanism similar to Xinu's ports [Comer84]. They provide a way for processes to pass arbitrary data and synchronize. A queue is created dynamically and is identified by an arbitrary (but unique) name that is global to all processes. Xsim provides functions for creating and destroying queues, enqueueing items, dequeueing items, scanning enqueued items, and controlling the operation of a queue (*e.g.*, setting a maximum queue length). Statistics on queues are maintained automatically.

A simulation program consists of one or more Xsim processes. These processes normally, but need not, represent servers. A server usually has one or more queues associated with it. For example, a single server simulation might have a *producer* process that simply puts items on a queue while a *consumer* process removes items from the queue (see Figure 6.1). The consumer blocks if it tries to dequeue an item when the queue is empty. The producer waits for some period of time to elapse and then enqueues an item on the consumer's queue, causing the consumer to resume execution if it had been blocked. Arbitrary data items are put on the queue; the queue management routines simply copy the data. In this example, items placed on the queue might include the value of the simulation clock at the time the item was enqueued and the amount of service time the item requires (*i.e.*, how long the consumer process should wait once the item has been dequeued). A simulation

continues until either the simulation clock exceeds some specified value, a specified real time period elapses, the program is interrupted, or a function is explicitly called to terminate the simulation.

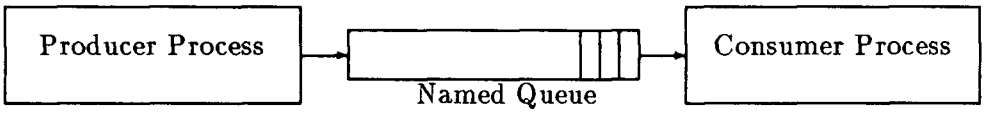

Figure 6.1: A Single Server Model

A more complete description of Xsim can be found in its user's manual [Brachman88b].

6.2 Methodology

This section discusses the simulation model and parameters used in the experiments. Following a description of elements common to all simulation runs, unique characteristics of the two basic configurations simulated are discussed. Next, the network topologies and simulation parameters are described.

In the first series of simulations, the objective was to compare end-to-end message throughput of the message stream based method using recipient buffer space reservations (the *fragmenting approach*) with a scheme performing neither fragmentation nor buffer reservations (the *non-fragmenting approach*). The effect of increasing recipient MTA congestion on the throughput of large messages was measured. Results for the throughput of single fragment messages are not presented because the recipient buffer space allocation scheme introduces many more single fragment messages and its throughput of single fragment messages is consistently higher, distorting the performance of the larger messages. The increased throughput of single fragment messages in the fragmenting system is

also a consequence of the system's greater degree of fairness. The hypothesis is that the non-fragmenting approach should not perform as well since congestion at a recipient MTA should spread and adversely affect other transfers. Many singleton simulations were run to observe the effect of changes to message stream parameters.

In the second series of simulations, the objective was to estimate the sensitivity of the results of the first series to workload. Toward this end, a configuration was rerun while varying the interarrival time of messages.

The simulation model uses the Xsim library and consists of about 5000 lines of C code. The simulations were performed on Sun 3 computers running SunOS version 3.2, 3.5, or 4.0.

6.2.1 Common Simulation Elements

In every simulation, an MTA comprises several processes, a fixed total number of buffers, input parameters, a routing strategy, and a buffer management scheme. The internal representation of the network is constructed from a specification of network topology read from a file. A simulation run creates a record of its initial state, input parameters, and, upon completion, a detailed summary. Statistics and state information are also generated every 4000 simulation seconds. Each run uses a different sequence of random numbers since the generator[1] is initialized using the system clock. Simulation begins with the queues of each MTA empty and continues until a steady state is achieved.

At the beginning of a simulation, a "parent" MTA process is created for each MTA in the network. Each parent MTA process generates one sender and one receiver process for each link to an adjacent MTA and then simulates the arrival of new messages into the

[1] The Berkeley Unix `random()` function, a non-linear additive feedback random number generator, was used. It has a period greater than 2^{69}.

system. A sender process waits for a message to arrive on the queue for the outgoing link it manages, negotiates with the receiver process on the other end of the link, and transfers the message if buffer space has been allocated by the receiver. Except for input messages that are rejected due to lack of input buffer space, messages are never discarded. Once accepted by the network, every message will eventually be delivered. Round-robin message scheduling is used.

A receiver process waits for the sender on the other end of its link to request buffer space. If space is available, the receiver accepts the message and places it in the queue of one of its MTA's sender processes according to the routing algorithm. If no space is available, the sender must retry later. Data can be transferred only in one direction at a time on a link. Multiple sender and receiver processes may be simultaneously active at a particular MTA; *i.e.*, an MTA can be either receiving or transmitting on each of its links simultaneously.

Since messages are submitted in their entirety in the message handling environment, buffer class zero must be large enough so that a reasonable number of messages, including some very large ones, can be held. Because adjusting the input buffer limit would discriminate against bigger messages more than smaller ones, the buffer class limit of class zero buffers is not adjusted. To prevent smaller messages from monopolizing input buffers, they are not permitted to use more than half of the input buffers.

For the first series of simulation experiments, the interarrival time of new messages at each MTA is normally distributed with a mean of 50 simulation seconds. Message sizes are drawn from a distribution based on the measurements presented in Section 3.7, with a minimum message size of one fragment and a maximum size of 100 fragments (Table 6.1). It was necessary to reduce the upper limit on the message size from the measured value to

make the simulations tractable. Messages within a particular range of sizes are chosen with equal probability. The recipient MTA for a newly created message is randomly chosen, with each MTA equally likely. An MTA may not send a message to itself. In each configuration, an MTA allocates 1000 buffers for input traffic, with each buffer consisting of 1000 bytes. Other buffer classes are initially allocated 100 buffers each so that the non-fragmenting configuration can transfer the largest possible message. A new message is rejected if there is no input buffer space for it in buffer class zero.

Message Sizes (fragments)	Probability
1	.70
2–30	.20
31–100	.10

Table 6.1: Distribution of Message Sizes

The simple hop-count scheme is used for determining the buffer class in which a fragment should be placed; *i.e.*, a fragment having made N hops must be placed in buffer class N. Each MTA has as many buffer classes as there are MTAs in the network less one.

Of the 500 buffers allocated to the recipient buffer space pool at each MTA, 25% are dedicated to single fragment messages and 75% to multifragment messages. This prevents either class of messages from monopolizing the buffers. An MTA process creates a process to simulate the removal of messages from the recipient buffer pool. The time between successive removal events is selected from a normal distribution with a mean of 50 simulation seconds. The amount restored to the pool at each event is taken from the message size distribution.

To better evaluate the benefits of recipient buffer space preallocation, congestion at recipient MTAs is augmented by adding extra delay before some requests for recipient

buffer space are granted. This models an interval during which a message requiring recipient buffer space is blocked because there is insufficient space in the recipient buffer pool to store the entire message. Because there are no statistics available on how long messages remain in the recipient buffer pool in a real messaging system, a range of delays is used.

In the fragmenting system, only a control message requesting recipient buffer space (a *request message*) is subject to this *recipient buffer space allocation delay* (RBSAD). The control message granting recipient buffer space (the *grant message*), sent in response to a request message, and message stream transfers have their space preallocated and do not experience this delay. In the non-fragmenting system, no preallocation is performed and all messages are subject to the delay. The forwarding of a message from its next-to-last MTA to its recipient MTA triggers this delay with probability 0.1 in the simulation studies. Should the delay occur, the message remains at its next-to-last MTA for the duration of the delay. Since requests for recipient buffer space are handled first-come, first-served, no transfer requiring recipient buffer space will be accepted until the delay expires; other transfers passing through the MTA are processed normally. The RBSAD is randomly selected from a normal distribution with mean 50, 100, 200, 400, 800, or 1600 simulation seconds for a particular simulation. Simulations were also performed with the augmented delay turned off (denoted as RBSAD 0). Some topologies were not simulated at the higher RBSAD values because of the long simulation time required.

Throughput is based on end-to-end delivery of complete messages rather than on raw fragment throughput. In the message handling environment, this is the throughput with which the user is concerned. Transport of a message from originator to recipient only contributes to the total system throughput when the last fragment of the message is delivered.

The model makes several simplifications to reduce the complexity or execution time

of a simulation. Multiple recipient messages are not simulated, communication links are assumed to be reliable, and processing time is assumed to be negligible. The simple hop-count technique is used for accessing buffer classes instead of one of the more elaborate techniques mentioned in Section 4.1. Message transfer is half-duplex on a link and input buffer limits are not adjusted. Static routing is used. Many potentially interesting simulation parameters are not varied, primarily because the networks would take considerably more time to simulate. For example, the number of MTAs in a particular topology could have been varied or the number of transit buffers per MTA could have been increased. There are an endless number of performance statistics that could be gathered (*e.g.*, buffer utilization) but which would place additional demands on the memory requirements of the simulations.

Like most studies of this nature, confidence in the correct operation of the model is based on an analysis of tracing output produced by the simulation, placing many consistency checks in the simulation code, running auxiliary simulations with different parameters to verify trends, and finding consistent explanations for the results.

6.2.2 The Fragmenting Approach

The fragmenting approach uses message streams and recipient buffer space reservations. The storage requirement of a single fragment is one buffer. A message stream is always used for messages consisting of three or more fragments and is established incrementally as each successive MTA receives the first fragment of a message. When a new multifragment message is generated, it requires an extra buffer that is preallocated for the grant message that will be sent in response to the reservation request message. Request messages are not given special treatment and also require recipient buffer space. As each fragment is forwarded, its space in the input buffer pool is released by the originating MTA. Transfers

that do not use a message stream do not use reservations; they are not accepted at the recipient MTA until there is enough space in the recipient buffer pool.

Senders fragment a message according to simulation parameters and negotiation with the receiver process. For most simulation runs a message stream was allocated a single buffer for its lifetime at an MTA. In several auxiliary runs, multiple buffers were allocated to message streams. In this case, the actual number of transit buffers allocated to a message stream is the minimum of the total message size and the maximum message stream allocation. The allocation is retained until the message stream is completed at the MTA. The link scheduling algorithm allows a message stream to transfer at most one half of the allocation at a time. This enables an MTA to simultaneously forward and receive fragments belonging to a particular message stream and also provides fairer sharing of the link. As a consequence, a message not larger than one half of the maximum message stream allocation would not be fragmented. A message stream cannot be established at an intermediate MTA until all of the message stream allocation can be granted. This restriction was made to simplify the simulation and probably would not be present in an implementation of the system. Lifting the restriction would likely increase throughput and decrease delay since buffer utilization would be higher. For example, the message stream could be established and proceed normally using a smaller allocation initially. The allocation could be increased should transit buffers of the appropriate buffer class become available.

Simulation of recipient buffer space reservations involves sending a request to the recipient MTA for allocation of space to contain the entire message. The request is piggybacked onto the first fragment of the message. If there is space for the entire transfer, it is set aside and a grant message is returned to the originator. In this case the first fragment is not

resent and, if the original message was two fragments in length, the remaining fragment does not use a message stream.[2] If there is insufficient space, the recipient MTA queues the request and sends a grant to the recipient once space becomes available. Requests are serviced first-come, first-serve so that all requests will eventually be processed.

An originator begins a multifragment transfer when it receives a grant message from the recipient. If there is insufficient space at the time the recipient MTA receives a request, the first fragment of the message must be resent. Owing to their smaller size, the transfer time of a control message is normally much smaller than that of other single fragment messages and is set at 20% of the normal transfer time in the simulation experiments. For simplicity, however, control messages are assigned the buffer space of an entire fragment. This will tend to reduce the performance of message streams over a more accurate simulation.

The dynamic buffer class adjustment scheme proposed in Chapter 5 was implemented and used with message streams. Buffer class limits are recalculated and reassigned after every 50 buffer release events. A buffer may also be reassigned after a release event. The parameters controlling buffer class adjustment were selected based on observations made during development of the simulation and were not optimized.

Buffer classes do not prevent traffic flow imbalances that lead to traffic flowing in one direction monopolizing all the buffers of a certain buffer class. This can result in reduced throughput and unfairness unless precautions are taken. This problem of buffer monopolization was partially addressed by a mechanism tying in with the buffer class adjustment strategy. Once outgoing link usage within a buffer class exceeds a limit, the buffer class is marked for expansion. Each outgoing link is given a minimum allocation out of the buffer class limit and, if a link uses more than 80% of the remaining buffer class

[2] This design decision was made early on to simplify the implementation of the simulation. A message stream could be used to transfer single fragment messages.

limit, the buffer class is marked for expansion. A buffer class is also marked for expansion when there are insufficient buffers in the buffer class to grant a request.

6.2.3 The Non-Fragmenting Approach

Although the non-fragmenting approach is meant to model current systems, which typically use neither buffer classes nor recipient buffer space preallocation, preliminary simulations experienced S/F deadlock. The simplest means of overcoming the deadlock was to introduce buffer classes and to use fixed buffer class limits that would allow the largest message to be transferred.

The non-fragmenting approach does not use message streams, dynamic buffer class adjustment, or recipient buffer space preallocation. Messages are transmitted in their entirety at each hop. Enough transit buffer space to hold the entire message must be available at each intermediate MTA and recipient buffer space to hold the entire message must be available before a message can be forwarded from its next-to-last hop to its recipient MTA. Requests for transit buffers and recipient buffer space are handled first-come, first-served.

6.2.4 Network Topologies

Four network topologies were studied in the simulations:

1. The *unidirectional ring* topology limits traffic flow to a clockwise direction with each MTA having a single incoming link and a single outgoing link. In this topology, traffic load is distributed evenly among 20 MTAs.

2. The *bidirectional ring* topology also comprises 20 MTAs. The routing algorithm chooses the shortest route or, if the recipient MTA is the same distance in either direction, a route is chosen at random.

3. The *linear array* topology consists of a string of 20 MTAs. In addition to its simplicity, this topology imposes heavier loads on MTAs closer to the middle of the string as well as offering some very long paths. While probably not a realistic configuration, it might be seen in a sequence of bridges or gateways.

4. The simplified *CDNnet* topology (Figure 6.2) models a more realistic topology for a message handling system than do the others. The mean number of hops per transfer is smaller than in the other topologies and the distribution of traffic load is less uniform. Arbitrary but minimal or near minimal length routes were selected for each originator and recipient pair. Because the complete CDNnet topology consists of too many MTAs and links to be simulated in a reasonable amount of time, a subset of 15 MTAs was chosen.

6.2.5 Simulation Parameters

The major simulation parameters are summarized in Table 6.2.

Parameter	Value
Simulation Duration	64000 simulated seconds
Fragment Size	1000 bytes (1 buffer)
Input Buffer Size	1000 Fragments
Data Fragment Transmission Time (t)	1 sec. and 2 secs. per fragment
Control Message Transmission Time	$t/5$ secs. per message
Message Stream Allocation	1 buffer
Buffer Readjustment Counter	50
Minimum Buffer Class Allocation	4
Recipient Buffer Pool Size	500 Fragments
Recipient Buffer Removal Rate	0.02 events/sec.

Table 6.2: Major Simulation Parameters

The duration of the simulation was made long enough that the variation in mean

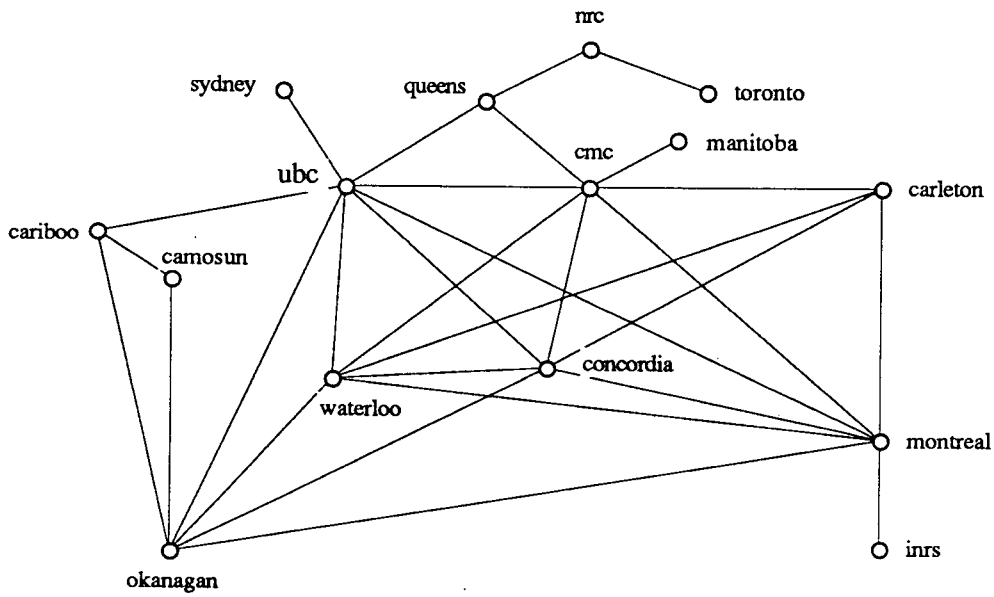

Figure 6.2: CDNnet Subset Topology

throughput over successive measurements was approximately 1%. Several simulations were rerun at random using a different stream of random numbers. The difference in mean throughput was typically about 2%, and none exceeded 5%.

The simulation runs with the data fragment transmission time (t) equal to 2.0 seconds were run first. The unexpected results obtained for the unidirectional topology prompted a repetition of these experiments at double the transmission speed (*i.e.*, $t = 1.0$ second). All other parameters were held constant. Given that a fragment represents 1000 bytes, $t = 2.0$ seconds is approximately equivalent to a transmission rate of 4800 bits per second. It is assumed that the propagation delay is negligible compared to the transmission speed and that the time to transmit a message (or message fragment) is linearly proportional to the size of the message.

In the second series of simulation experiments, the bidirectional ring was studied, varying only the mean message interarrival time. The RBSAD was set at 100. The choice of topology and RBSAD for the second series was based solely on the ability to do the simulations in a reasonable amount of time.

The real time to perform a simulation varied from 3 hours to over 4 days.

6.3 Results

The results of the simulation experiments are presented and discussed in this section. Simulations of the four topologies were performed for the fragmenting and non-fragmenting systems. In the first set of experiments, both the RBSAD and the transmission speed were varied. Measurements of the mean throughput of large transfers (*i.e.*, transfers involving messages two or more fragments in length) versus the mean RBSAD are plotted for both transmission speeds. For the CDNnet topology, the effects of the message stream allo-

cation on delay and throughput were measured, as was the effect of transmission speed on throughput. In the second set of experiments, a bidirectional ring configuration was simulated while varying the interarrival time of messages. An RBSAD of 100, representing a relatively heavy workload, was used for each of these runs. Measurements of the mean throughput of large transfers versus the mean interarrival time of messages are plotted.

An initial observation is that the large number of transit buffers necessitated by a fair comparison with the non-fragmenting approach limited the influence of dynamic adjustment to those runs of the linear and CDNnet topologies with higher congestion. The other topologies distributed the network load more evenly and since buffer class limits were not reached they were not eligible for adjustment.

6.3.1 Unidirectional Ring

Results for the unidirectional ring topology are plotted in Figure 6.3. This is the only topology where reservations did not outperform non-reservations through the majority of simulation runs at $t = 2.0$. Review of the simulation results led to three observations and hypotheses to account for the relatively poor performance of the reservation scheme. The hypotheses are similar in that they assert that the poorer performance is emphasized by the more limited connectivity of the network.

The first observation is that a transfer suffers a larger delay before starting than in other topologies since a reservation request and the corresponding grant message must traverse a total distance equal to the number of MTAs in the network, much longer than is the case for any of the other topologies. Since performance of message stream transfers is related to the delay incurred in the delivery of control messages, decreasing this delay might increase the throughput of the system. To evaluate the influence of the control message transfer

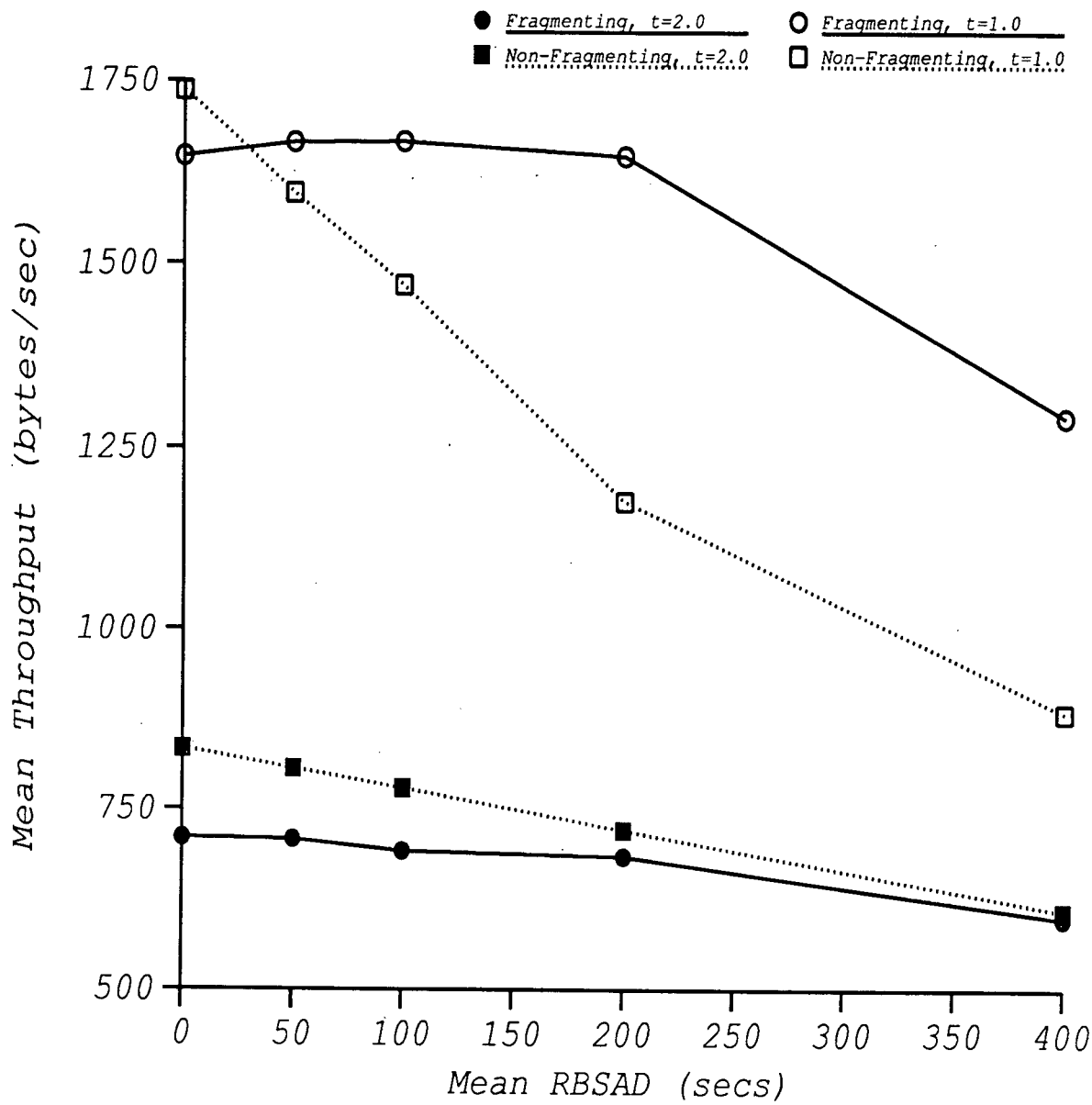

Figure 6.3: Unidirectional Ring: Throughput *vs.* RBSAD

speed, two runs were performed with $t = 2.0$ and an RBSAD of 0. In the first run, the control message transmission speed was doubled to $t/10.0$ and in the second run it was increased to $t/50.0$. In the former run, throughput did not change. In the latter run, it increased a modest 5%. It may be inferred from this that the time to transmit a control message is not a significant throughput bottleneck (over the entire length of a simulation).

The second observation is that the mean number of reservation requests waiting to be serviced at an MTA is much lower than in the other topologies. This suggests that request messages are taking longer to be delivered to the recipient MTA. As increasing the transmission speed of control messages was ineffectual (and did not increase the length of the queue of reservation requests), reduction of the queueing delay of a control message by scheduling it more favourably was evaluated. Three extra simulation runs were made (described below) to look at the effect of giving control messages a higher scheduling priority than data messages. These runs also did not result in significant changes in system throughput. The conclusion to be reached from these additional experiments is that giving preferential treatment to control messages does not help throughput in this system.

The last and most significant observation is that the lengths of transit queues at intermediate MTAs are observed to be much longer than what is found when fragmentation is used in other topologies. For example, the length of the single queue at an MTA in the unidirectional ring at RBSAD 0 is approximately 2.5 times longer than either of the queues in the corresponding bidirectional ring configuration. Also, the total number of transit buffers in use at the former is about 2.5 times the number at the latter. The reason for these differences is that there is only one outgoing link at each MTA in the unidirectional topology. Because of this queueing delay, very large transfers took considerably longer to complete than in the non-fragmenting system, often much more than one measurement

interval (4000 simulation seconds). The other topologies have multiple, simultaneously active outgoing links that results in shorter queue lengths at each link. Queue lengths in the non-fragmenting, unidirectional case are much shorter because some transfers use more than one transit buffer.

The preceding observations prompted auxiliary simulation runs to evaluate two methods of reducing queue lengths. The first method was to decrease the mean number of active message streams at each intermediate MTA (thereby decreasing the mean queue length) by increasing the maximum message stream allocation. When a message stream has an allocation of more than one buffer, fragments may be delivered with lower delay because of the increased parallelism. If a message stream is assigned only a single buffer at each hop along its route for the duration of the transfer, as was the case for the standard runs in the first series of simulation experiments, an MTA cannot simultaneously receive fragment $N + 1$ and send fragment N belonging to the same message stream. A larger maximum allocation has a detrimental effect, however, in that the queueing delay experienced by smaller transfers will increase.

Simulations were run with an RBSAD of 0 using maximum message stream allocations of 4, 8, 16, 32, 64, and 80 buffers. There was no significant change in throughput in any of the runs. Transit queue lengths decreased significantly only when 16 or more buffers were allocated. As mentioned earlier, increased message stream allocations should result in increased queueing delays for smaller transfers. Because the simulations did not keep track of the delivery delay for different traffic groups in these simulations, this queueing delay must be inferred from two observations. The first is that the throughput of single fragment messages decreased as the message stream allocation increased. The second is that the percentage of new single fragment messages rejected owing to insufficient input

buffer space increased as the message stream allocation increased.

Some of these simulations were rerun with an RBSAD of 100 using maximum message stream allocations of 4, 8, and 16 buffers. One set of these runs was performed giving a high priority to all control messages; control messages were serviced first-come, first-served, before data messages. There was no significant change in throughput in any of these runs, again indicating that increasing the message stream allocation or giving control messages preferential treatment does not improve system throughput appreciably.

The second method used to decrease queue lengths was to increase the message transmission speed. Consequently, a complete set of simulation runs was performed with $t = 1.0$ (Figure 6.3) and simulation runs of the other topologies were performed for both $t = 2.0$ and $t = 1.0$. The results show that increasing the transmission speed increases the relative throughput performance of fragmentation over non-fragmentation compared to runs with $t = 2.0$. This is because increasing transmission speed is more beneficial to fragmented transfers since their recipient buffer space is preallocated. Congestion caused by lack of recipient buffer space prevents non-fragmented transfers from taking full advantage of the increased speed. Additional runs were performed at RBSAD 100 with buffer allocations of 4 and 8 and with one run limiting the number of active message streams at an originating MTA to 10 (1/3 to 1/4 of the usual number for this configuration). There was no significant change in throughput in any of these additional runs. The conclusions to be drawn from these observations is that the relative throughput performance of fragmentation over non-fragmentation increases as transmission speed increases and that the performance of fragmentation is more sensitive to transmission speed than to message stream buffer allocations.

6.3.2 Bidirectional Ring

Results for the bidirectional ring topology are plotted in Figure 6.4.

Fragmentation outperforms non-fragmentation up to the heaviest RBSAD at both transmission speeds. While the throughput of the fragmenting system is nearly constant up until moderate congestion, throughput of the non-fragmenting system declines rapidly. For both transmission speeds at RBSAD 400 the throughput of the fragmenting system is about double that of the non-fragmenting system.

In the non-fragmenting system, throughput of the bidirectional ring at $t = 2.0$ deteriorates more quickly than in the unidirectional ring topology. This is likely due to the increased demand for transit buffers in the bidirectional ring. In a unidirectional ring, a transfer from a particular originating MTA shares the buffer class at each hop only with other transfers originating at the same MTA.

6.3.3 Linear Array

Results for the linear array topology are plotted in Figure 6.5.[3] At $t = 2.0$, fragmentation performs as much as 45% better and at $t = 1.0$ almost twice as well at RBSAD 400. The improvement of fragmentation over non-fragmentation when the transmission speed is doubled is observed to be more than double.

6.3.4 CDNnet Subset

Results for the CDNnet subset topology are plotted in Figure 6.6. Several interesting trends appear in the data.

[3] The measurement of the fragmenting system at (800, 583) for $t = 1.0$ is the mean of 12 observations instead of the usual 16.

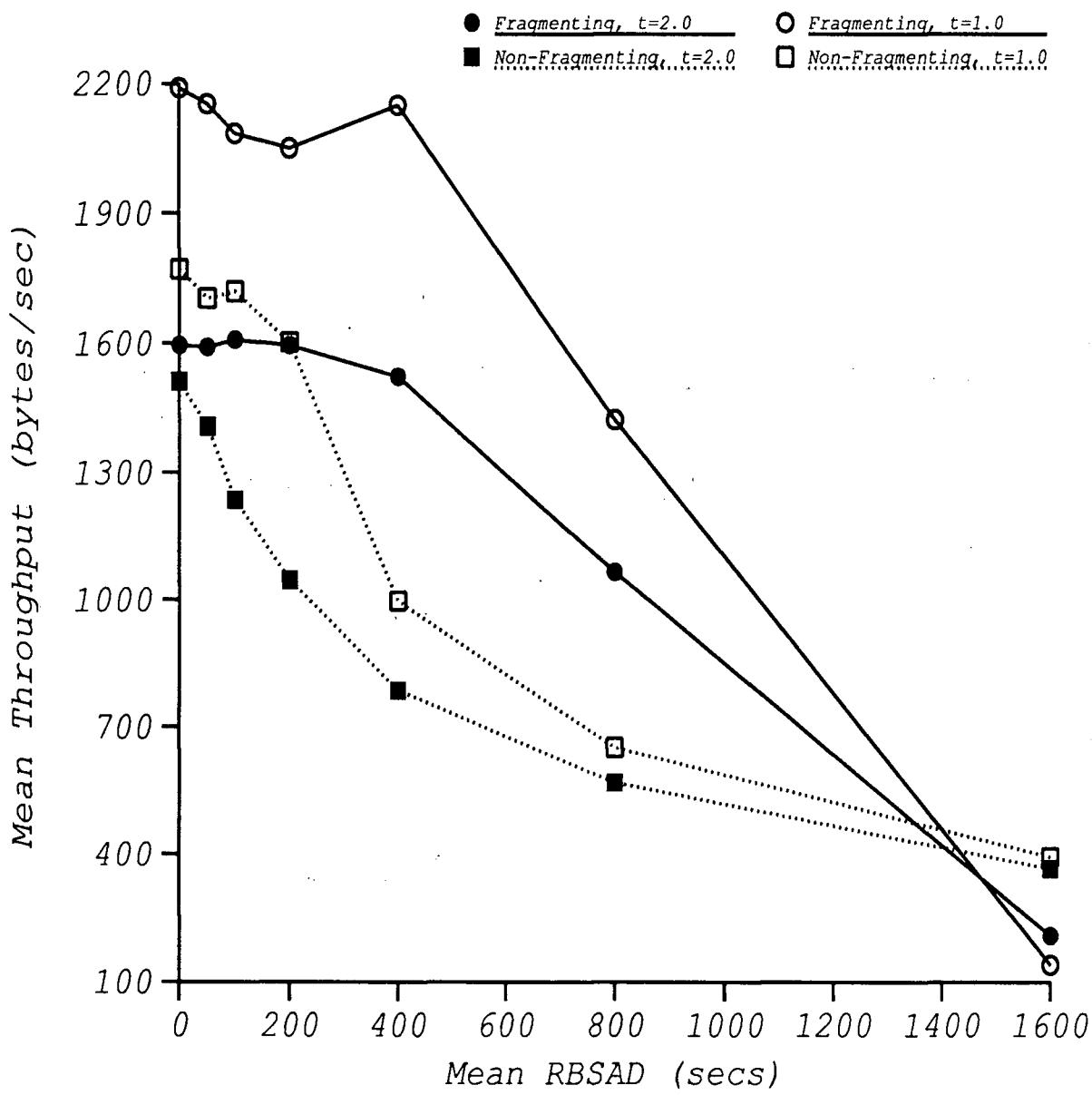

Figure 6.4: Bidirectional Ring: Throughput *vs.* RBSAD

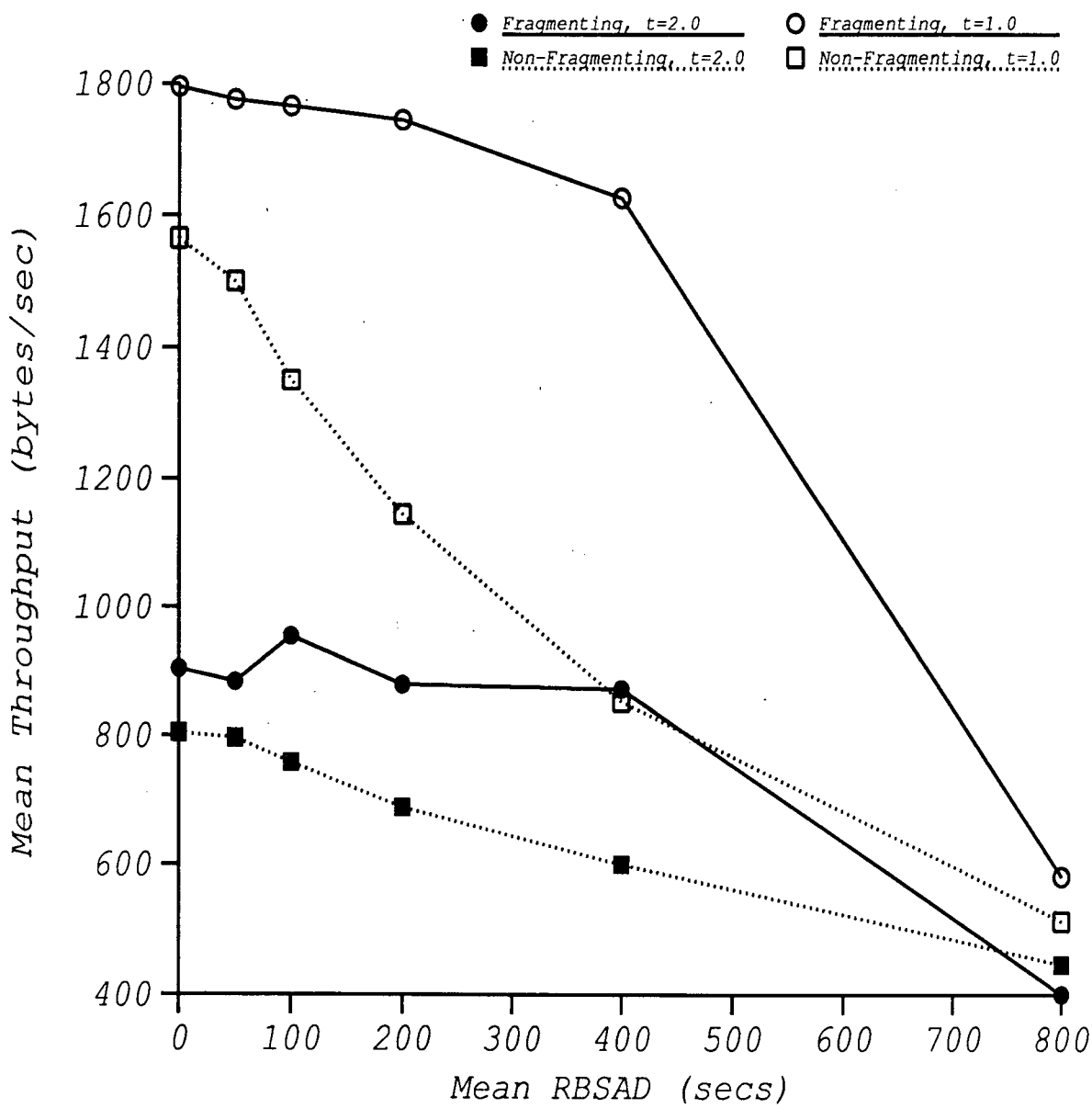

Figure 6.5: Linear Array: Throughput *vs.* RBSAD

Fragmentation outperforms non-fragmentation by a large margin at both transmission speeds through the entire range of congestion.

Perhaps unexpectedly, doubling the transmission speed does not make a significant difference in the performance of either system. This phenomenon can be attributed to the relatively low mean number of hops a message makes as a result of the highly interconnected topology. Thus, the bottleneck in the system becomes the size of the recipient buffer pool and increasing transmission speed has little effect.

Another unusual result is that the throughput of the non-fragmenting system at $t = 1.0$ declines relative to the same system with $t = 2.0$ for the first three data points and increases for the others. This behaviour can probably also be attributed to the highly interconnected topology as well as the uneven distribution of congestion. One possible explanation is that at lower RBSAD values the increased transmission speed increases demand for recipient buffer space and the resulting congestion in the transit buffer system decreases system throughput. As the RBSAD increases, demand for recipient buffer space decreases and system throughput increases. Therefore, the observed throughput may be a result of the interaction of the two types of congestion. Also, the third data point for $t = 1.0$ is somewhat lower than expected; there was an unusually low measurement recorded in the results for that simulation run.

Figure 6.7 shows how transmission speed affects system throughput in the fragmenting and non-fragmenting systems. The RBSAD was fixed at 100.0 seconds for these runs. Fragmentation outperforms non-fragmentation through the range of transmission speeds. Because demand for recipient buffer space becomes the bottleneck in this configuration, throughput does not increase as much as might be expected when the transmission speed is increased.

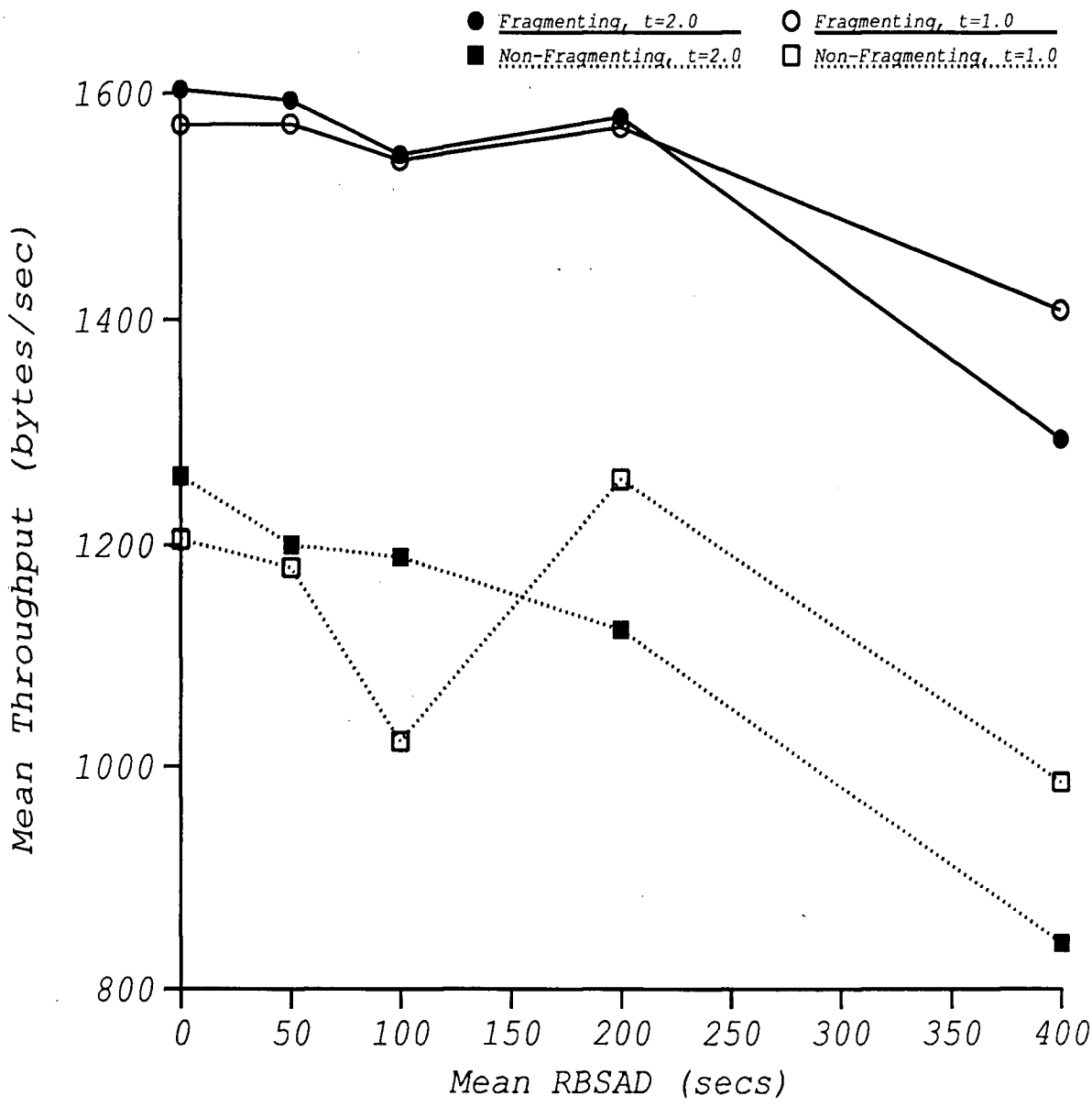

Figure 6.6: CDNnet Subset: Throughput *vs.* RBSAD

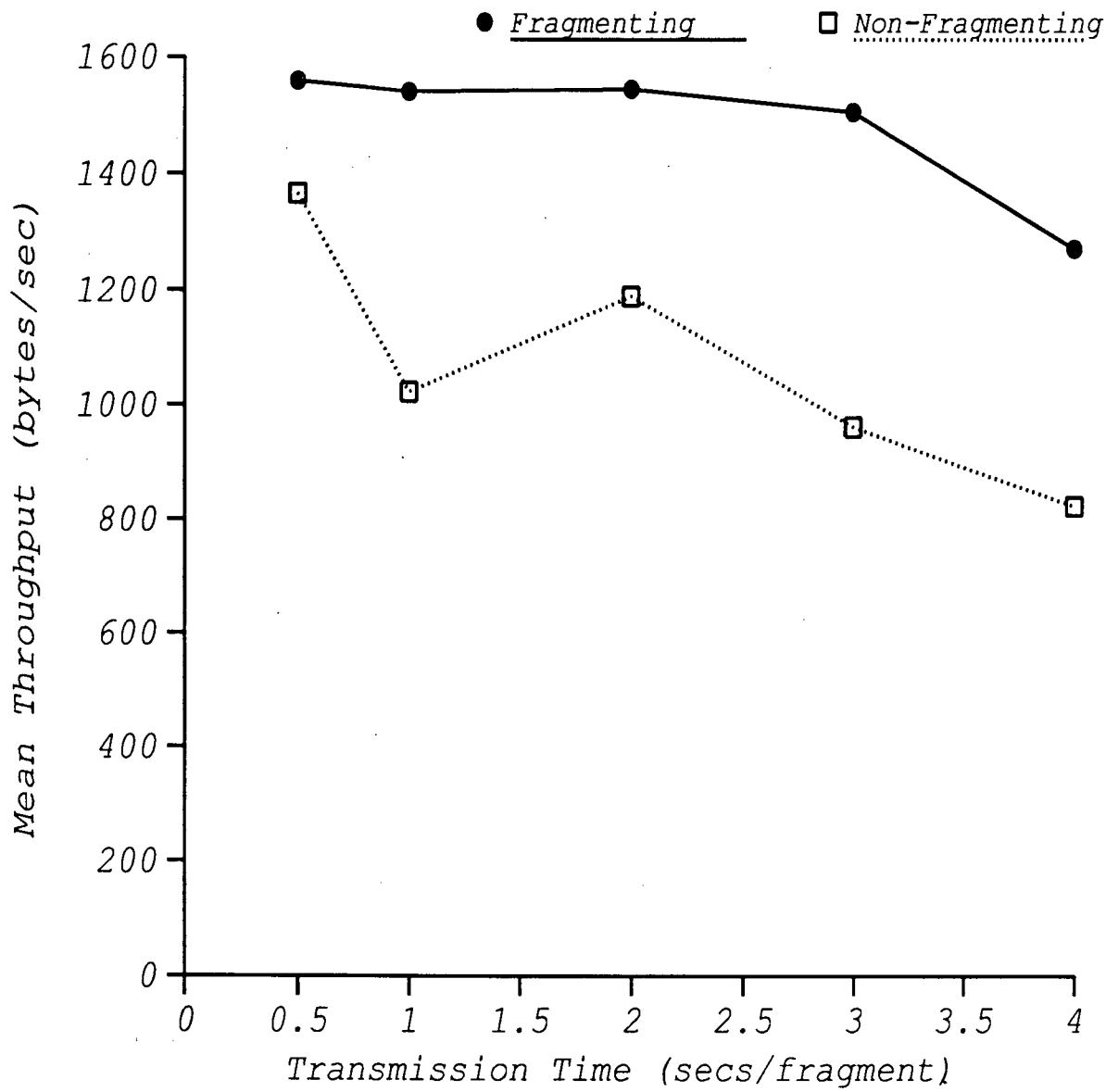

Figure 6.7: CDNnet Subset: Throughput *vs.* Transmission Time

Simulations were performed to examine the influence of the message stream allocation on the total time required to deliver a message to its recipient MTA (Figure 6.8). All runs were performed with $t = 2.0$ and the RBSAD at 100.0 seconds, which represents a heavy load on the network. Results are shown for the fragmenting system with message stream allocations of 1, 16, 32, and 64 transit buffers.[4] Results for the non-fragmenting system are also plotted. Delay was measured for messages one fragment in length, messages between 2 and 10 fragments in length, 11 to 20 fragments in length, and so on.

Figure 6.8 illustrates how fragmentation greatly reduces the mean delay experienced by smaller transfers at the expense of larger transfers. At the same time, the throughput rate for the fragmenting system is uniformly higher. Fragmentation is shown to be fairer in that smaller messages incur less delay than larger messages. As message stream allocation increases, delay for larger transfers approaches that of the non-fragmenting system.

In the non-fragmenting system, messages smaller than about 40 fragments in size take much longer to be delivered than in the fragmenting system. This is because small messages may have to wait for many larger messages to be forwarded before they are serviced. In the fragmenting system, small messages do not wait nearly as long to get forwarded. On the other hand, large messages do not do as well in the fragmenting system. Because round-robin scheduling is used, they must wait for all other ready message streams and single fragment messages to be serviced before they obtain their "quantum" of service. It should be noted that because the message size distribution is heavily slanted towards smaller messages, the delay results for the higher ranges of message sizes (messages of size 40 fragments and greater) are less accurate as there are fewer such messages.

[4] Message stream allocations of 2, 4, and 8 were simulated but are not plotted in the interest of clarity. The results follow the trend established by the data plotted in Figure 6.8. Note that for message stream allocations greater than 1, the effective maximum fragment size is one half of the message stream allocation.

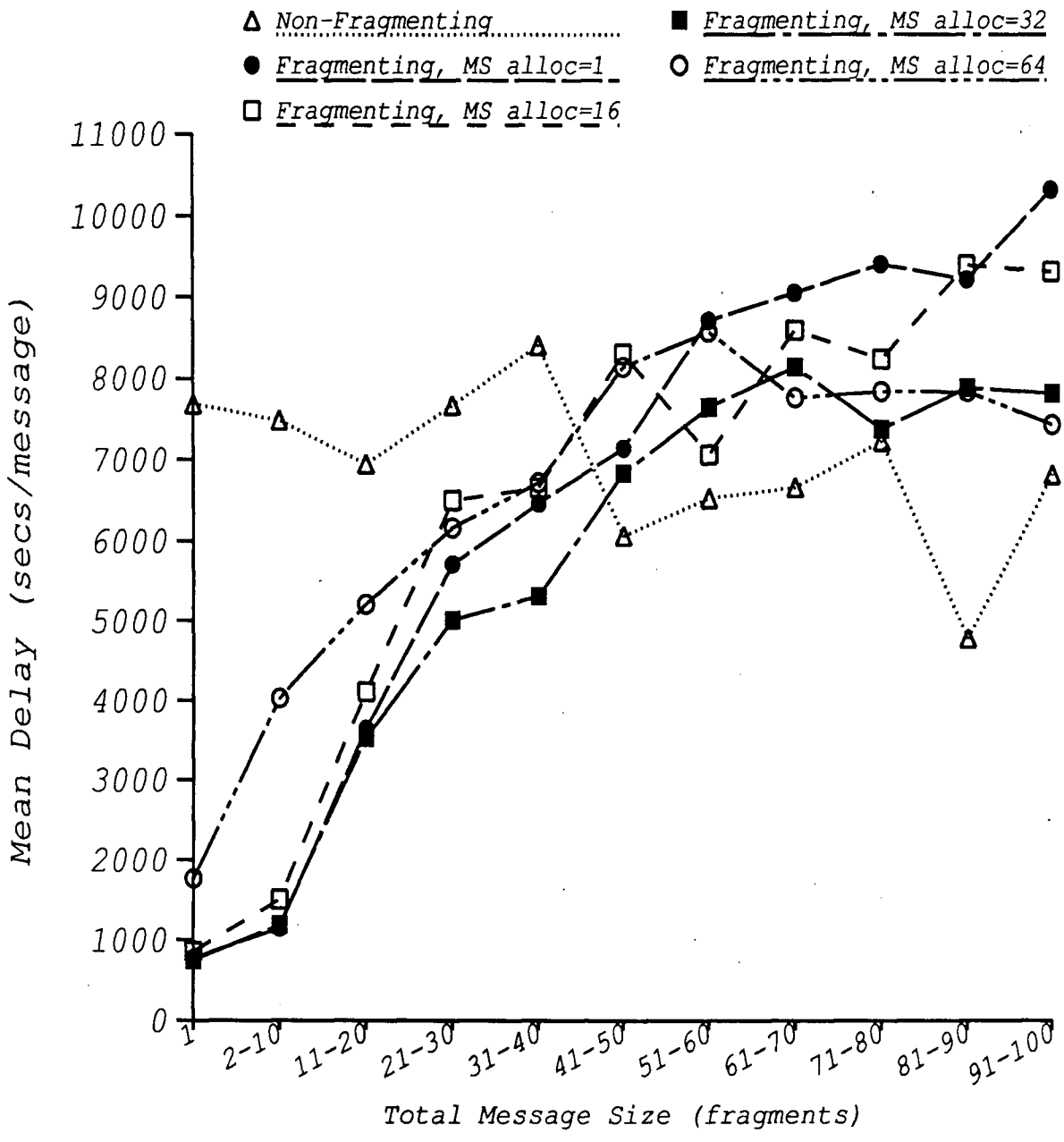

Figure 6.8: CDNnet Subset: Delay *vs.* Message Size

The results of simulations that varied the fragment size from 1k to 64k bytes while keeping the transmission time at $t = 2.0$ and RBSAD at 100.0 seconds are plotted in Figure 6.9. The mean throughput rate of large messages decreases slightly as the fragment size increases, indicating that overall system throughput is relatively insensitive to the fragment size up to some limit and, therefore, also relatively insensitive to the message stream allocation.

6.3.5 Sensitivity to Workload

To estimate the degree to which the results depend on the network load, a second set of simulation experiments were undertaken. Simulations of the bidirectional ring were run using an RBSAD of 100 and with mean interarrival times of 10, 25, 50, 75, 100, 150, 200, 400, and 800 seconds while all other parameters were held constant. The results are plotted in Figure 6.10.

At the shortest interarrival time, about 38% and 69% of messages were rejected in the fragmenting and non-fragmenting systems, respectively. At the longest interarrival time no messages were rejected in either system.

The results show that fragmentation continues to offer better throughput through a wide range of interarrival times. As expected, at extremely low loads throughput performance of the two techniques is equal.

6.4 Summary

The simulation study revealed several interesting characteristics of fragmenting and non-fragmenting message transfer systems:

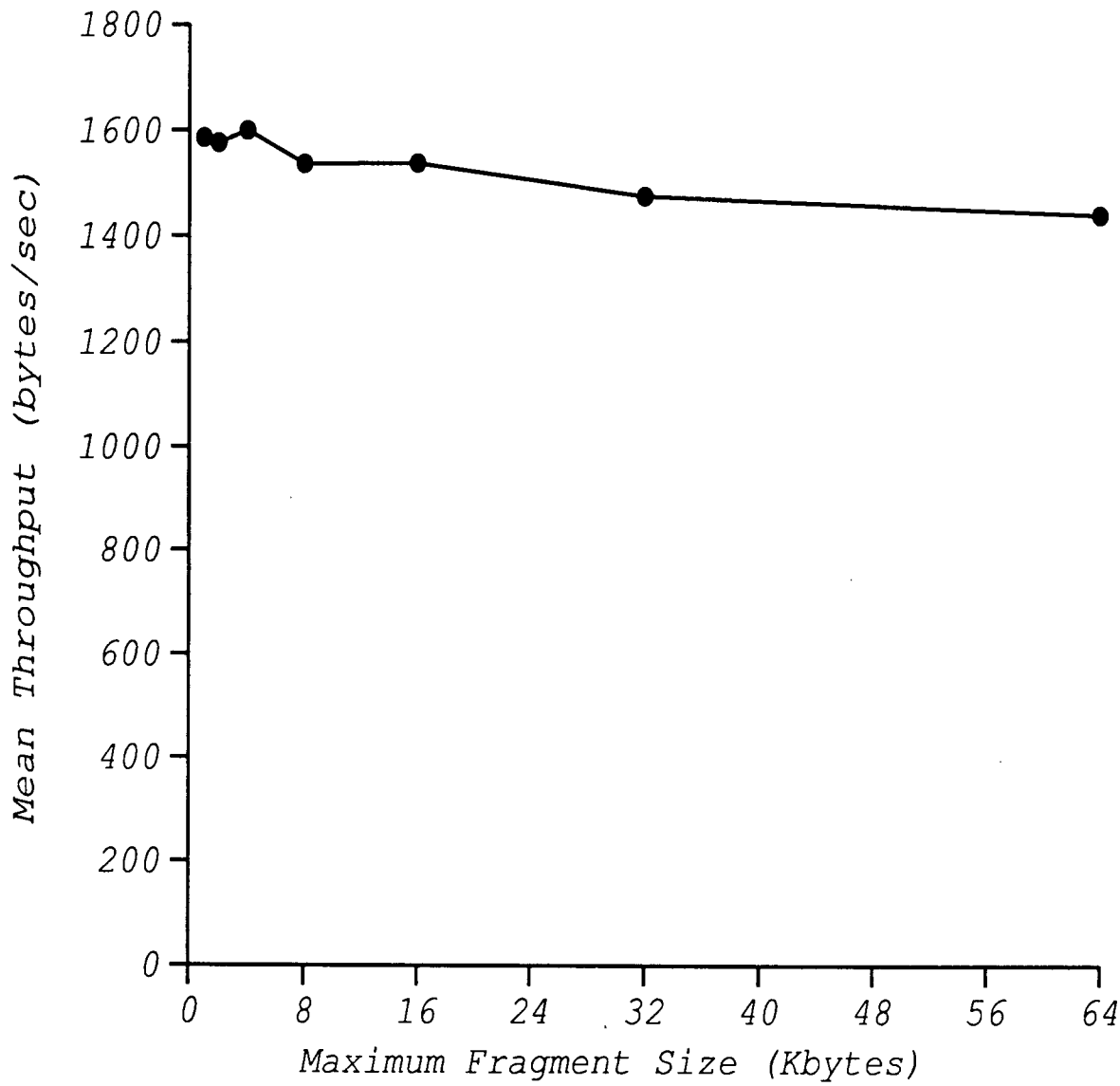

Figure 6.9: CDNnet Subset: Throughput *vs.* Fragment Size

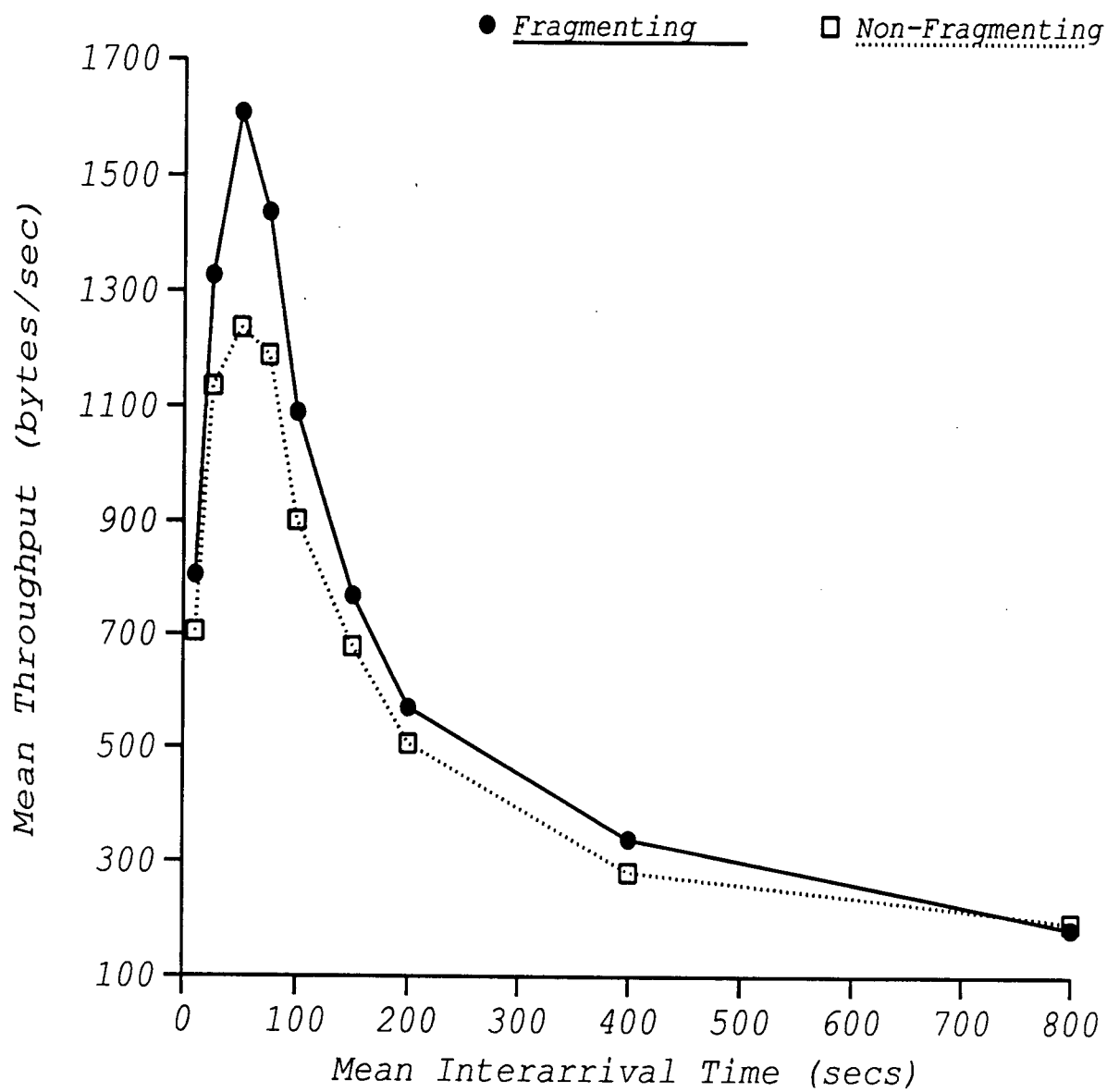

Figure 6.10: Bidirectional Ring: Throughput *vs.* Interarrival Time

- Fragmenting systems can avoid deadlock while non-fragmenting systems with the same resources will experience deadlock.

- Fragmenting systems offer significantly higher throughput than non-fragmenting systems over a wide range of network congestion. This superiority increases as transmission speeds increase and appears to be largely independent of the message interarrival rate. At some point demand for recipient buffer space becomes the system bottleneck and increasing transmission speed is of little benefit to system throughput. The point at which this occurs is dependent on network topology.

- The network throughput performance of message streams is not sensitive to the maximum message stream allocation over a wide range. Increasing the maximum allocation, at least up to the maximum value simulated (64k bytes), results in no significant change in the throughput of message stream transfers yet the throughput of single fragment messages decreases as a result of the decreased fairness. End-to-end delivery delay is much lower for smaller messages in a fragmenting system while delay for larger messages tends to be greater relative to the non-fragmenting system. The delivery delay for the largest messages can be reduced by increasing the message stream allocation. Giving preferential treatment to control messages does not always improve overall message stream throughput appreciably.

- Two important bottlenecks can be identified in message transport systems: transmission speed and the size of the recipient buffer pool.

Chapter 7

Conclusions

Starting from a characterization and investigation of message transfer systems, this thesis has developed a framework for integrating the resource management aspects of message transfer systems in a structured way. It is often true, however, that in the process of acquiring knowledge as many questions are raised as answered. This chapter looks at areas that deserve further study and concludes with a summary and evaluation of the thesis.

7.1 Summary and Evaluation

The major goal of this thesis has been to study application level, S/F message transfer systems, identify shortcomings of existing systems, and suggest improvements. There are four major contributions as a result.

A characterization of these systems has been given and deficiencies of popular existing systems have been identified. Although message transfer systems are proliferating and coming under increasingly heavy use, an investigation of message transfer systems has yet to appear in the literature. This study is long overdue and forms the basis of the following contributions.

A hierarchical deadlock prevention scheme was designed and shown to be correct. It is appropriate both for the message handling environment and for a reliable datagram environment. Existing systems, intended for lower networking levels, suffer many drawbacks that make them impractical for the message handling environment. The hierarchical approach offers several advantages over approaches appearing in the literature.

The third contribution is a study of design considerations for resource management components of message transfer systems. A framework whereby techniques to provide system components such as message fragmentation, message streams, congestion control, transit buffer management, and recipient buffer space allocation can be integrated were presented. Message transfer systems designed using this framework should offer superior performance and reliability over current systems.

Lastly, a simulation study of message streams and buffer management offered evidence as to the practicality of the proposals of Chapter 5.

As discussed in the next section, there are several interesting related research topics that could be pursued. The extensive literature review provides a good starting point for such research.

7.2 Future Work and Research Directions

The fact that there is no prototype implementation of a message transfer system based on the framework of Chapter 5 as part of the thesis requires some explanation. Many aspects of the framework presented in this thesis address performance related issues and evaluation of their efficacy is highly dependent upon the workload used. The manner in which tradeoffs between features, cost, performance parameters, and qualitative characteristics are resolved also influence the nature of the resulting system. Because the performance of

any single implementation is heavily dependent on the environment in which it is evaluated and the criteria that are used, many implementations in different environments would be required to make it possible to gather enough evidence to support more substantial claims regarding the framework. Also, it was deemed that a complete implementation was infeasible for the thesis and a partial implementation would cover only a very small subset of the concepts presented here. Instead, the simulation study presented in Chapter 6 provides many insights into performance aspects of message streams and the recipient buffer space preallocation strategy.

This thesis has addressed the task of *designing* some of the components of message transfer systems. The success of an implementation is often gauged relative to the existing, mostly *ad hoc* internetwork message handling systems. This thesis has pointed out the many deficiencies in current systems and has argued that the framework presented here overcomes many of these deficiencies. Nevertheless, valuable insights are often obtained while implementing systems and an implementation would surely aid in the refinement of some of the ideas presented in this thesis.

Some important aspects of message transfer systems have not been addressed in this thesis and would be required in a comprehensive framework. One such aspect is the important and difficult task of message routing. Security aspects of message transfer systems have not been addressed in this thesis and have received little attention in the literature, although some security issues are addressed in the X.400/1988 recommendations and the CCITT X.500 series of recommendations.[1]

An area of further investigation involves a study of the types of information that could be distributed to improve congestion control, determine message lifetimes, and so on. This

[1] The ISO variant of the X.500 recommendations is known as ISO 9594 (Information Processing Systems – Open Systems Interconnection – The Directory).

would require further simulation work or an implementation of a design. A more detailed analysis of existing message transfer systems could be a first step in such an investigation.

Other areas worthy of study are the reliability and fault tolerance of message transfer systems. The use of replication and mechanisms to recover from failures and insure data consistency are interesting issues that have not been adequately researched.

The simulation study could easily be extended to incorporate new algorithms and parameters. For example, other techniques for dynamically updating buffer allocations (*e.g.*, [Tipper88]) could be examined.

Scheduling message transfers subject to fairness, performance, and economic constraints is a difficult problem. There does not appear to be much literature on the type of prioritized message scheduling discussed in Section 5.2.2.

A study might be made of how congestion control mechanisms at different networking levels might cooperate to improve their effectiveness. At present, these mechanisms are largely autonomous and may very well interfere with each other.

Improved performance could be realized if a message transfer system could decide when it was appropriate to change the route of a message stream. Research into the relationship between message stream routing and congestion control would be a first step in this direction.

With an appropriate network level, message transfer systems could be used by real-time applications such as conferencing. Study of the special requirements of these applications with respect to the framework of Chapter 5 would be an interesting project.

There is a great deal of interest in message handling systems and store-and-forward message transfer, as is evidenced by the considerable standardization efforts put forth

recently and the growing number of academic and commercial implementations. The many favourable characteristics of message transfer systems and their increasing use should insure their continued development. Hopefully this thesis will contribute to a better understanding of store-and-forward message transfer and improved message handling systems.

References

[Aggarwal85] Aggarwal, S., Sabnani, K., and Gopinath, B. "A New File Transfer Protocol", *AT&T Technical Journal*, vol. 64, no. 10, (December 1985), pp. 2387-2411.

[Aggarwal86] Aggarwal, S., and Sabnani, K. "Formal Specification of a File Transfer Protocol", *Proc. IEEE INFOCOM '86*, 1986, pp. 47-57.

[Allman86] Allman, Eric. "SENDMAIL – An Internetwork Mail Router", *Unix System Manager's Manual*, SMM:16, Computer Systems Research Group, EECS, University of California at Berkeley, 1986.

[Błażewicz87a] Błażewicz, Jacek, Brzeziński, Jerzy, and Gambosi, Giorgio. "Time-Stamp Approach to Store-and-Forward Deadlock Prevention", *IEEE Trans. on Communications*, vol. COM-35, no. 5, (May 1987), pp. 490-495.

[Błażewicz87b] Błażewicz, Jacek, Brzeziński, Jerzy, and Gambosi, Giorgio. "Time-Stamp Approach to Prevention of Different Deadlock Types in Store-and-Forward Networks", *IEEE Trans. on Communications*, vol. COM-35, no. 5, (May 1987), pp. 564-566.

[Bartoli83] Bartoli, Paul D. "The Application Layer of the Reference Model of Open Systems Interconnection", *Proc. of the IEEE*, vol. 71, no. 12, (December 1983), pp. 1404-1407.

[Bennett82] Bennett, C. J. "The Overheads of Transnetwork Fragmentation", *Computer Networks*, vol. 6, 1982, pp. 21-36.

[Bharath-Kumar81] Bharath-Kumar, K., and Jaffe, J. M. "A New Approach to

Performance-Oriented Flow Control", *IEEE Trans. on Communications*, vol. COM-29, no. 4, (April 1981), pp. 427-435.

[Bongiovanni87] Bongiovanni, G., and Bovet, D. P. "Minimal Deadlock-Free Store-and-Forward Communication Networks", *Networks*, vol. 17, 1987, pp. 187-200.

[Brachman86] Brachman, Barry J. "EANft User's Guide", unpublished memorandum, University of British Columbia, Dept. of Computer Science, September 1986.

[Brachman87] Brachman, Barry J., and Chanson, Samuel T. "An Access Control Mechanism Based on the ISO FTAM Recommendation", *Proc. CIPS Congress '87*, May 1987, pp. 161-166.

[Brachman88a] Brachman, Barry J., and Chanson, Samuel T. "Fragmentation in Store-and-Forward Message Transfer", *IEEE Communications Magazine*, vol. 26, no. 7, (July 1988), pp. 18-27.

[Brachman88b] Brachman, Barry J. "Xsim – A Process-Based Discrete-Event Simulator", unpublished memorandum, University of British Columbia, Dept. of Computer Science, November 1988.

[Brachman89a] Brachman, Barry J., and Chanson, Samuel T. "Flow and Congestion Control in the Message Handling Environment", *Proc. IEEE INFOCOM '89*, 1989, pp. 721-730.

[Brachman89b] Brachman, Barry J., and Chanson, Samuel T. "A Simulation Study of Application Level Message Transfer Using Message Streams", in preparation.

[Brachman89c] Brachman, Barry J., and Chanson, Samuel T. "A Hierarchical Solution for Application Level Store-and-Forward Deadlock Prevention", submitted for publication.

[Case88] Case, Jeffrey D., Davin, James R., Fedor, Mark S., and Schoffstall, Martin L. "Introduction to the Simple Gateway Monitoring Protocol", *IEEE Network*, vol. 2, no. 2, (March 1988), pp. 43-49.

[CCITT88a] CCITT. "Draft Recommendation X.400: Message Handling Systems: System and Service Overview", Version 5.5, April 1988.

[CCITT88b] CCITT. "Draft Recommendation X.411: Message Handling Systems: Message Transfer System: Abstract Service Definition and Procedures", Version 6, March 1988.

[CCITT88c] CCITT. "Draft Recommendation X.413: Message Handling Systems: Message Store: Abstract Service Definition", Version 6, March 1988.

[CDNnet88] CDNnet Reports, no. 3, Fall 1988, CDNnet Headquarters, University of British Columbia.

[Cerf74] Cerf, Vinton G., and Kahn, Robert E. "A Protocol for Packet Network Interconnection", *IEEE Trans. on Communications*, vol. COM-22, no. 5, (May 1974), pp. 637-648.

[Cerf81] Cerf, Vinton G. "Packet Communication Technology", in *Protocols and Techniques for Data Communication Networks*, Franklin F. Kuo (ed.), Prentice-Hall, 1981.

[Chan87] Chan, Cheung-Wing, and Yum, Tak-Shing. "An Algorithm for Detecting and Resolving Store-and-Forward Deadlocks in Packet Switched Networks", *IEEE Trans. on Communications*, vol. COM-35, no. 8, (August 1987), pp. 801-807.

[Chandy83] Chandy, K. Mani, Misra, Jayadev, and Haas, Laura M. "Distributed Deadlock Detection", *ACM Trans. on Computer Systems*, vol. 1, no. 2, (May 1983), pp. 144-156.

[Chapin82] Chapin, A. Lyman. "Connectionless Data Transmission", *Computer Communication Review*, vol. 12, no. 2, (April 1982), pp. 21-61.

[Chapin83] Chapin, A. Lyman. "Connections and Connectionless Data Transmission", *Proc. of the IEEE*, vol. 71, no. 12, (December 1983), pp. 1365-1371.

[Cidon86a] Cidon, Israel, Jaffe, Jeffrey M., and Sidi, Moshe. "Local Distributed Deadlock Detection with Finite Buffers", *Proc. IEEE INFOCOM '86*, 1986, pp. 478-487.

[Cidon86b] Cidon, Israel, Jaffe, Jeffrey M., and Sidi, Moshe. "Local Distributed Deadlock Detection by Knot Detection", *Proc. ACM SIGCOMM*, (*Computer Communication Review*, vol. 16, no. 3), 1986, pp. 377-384.

[Clark87] Clark, David D., Lambert, Mark L., and Zhang, Lixia. "NETBLT: A High Throughput Transport Protocol", *Proc. ACM SIGCOMM*, (*Computer Communication Review*, vol. 17, no. 5), 1987, pp. 353-359.

[Clopper80] Clopper, Samuel E. "Features of the File Transfer Protocol and the Data Presentation Protocol", Report 4257, Bolt, Beranek and Newman, September 1980.

[Comer84] Comer, Douglas. Operating System Design, the Xinu Approach, Prentice-Hall, 1984.

[Crocker79] Crocker, David, Szurkowski, Edward S., and Farber, David J. "An Internetwork Memo Distribution Capability - MMDF", Dept. of Electrical Engineering, University of Delaware, September 1979.

[Cunningham83] Cunningham, Ian. "Message-Handling Systems and Protocols", *Proc. of the IEEE*, vol. 71, no. 12, (December 1983), pp. 1425-1430.

[Da Cruz84a] Da Cruz, Frank, and Catchings, Bill. "Kermit: A File-Transfer Protocol for Universities (Part 1)", *Byte*, vol. 9, no. 6, (June 1984), p. 255.

[Da Cruz84b] Da Cruz, Frank, and Catchings, Bill. "Kermit: A File-Transfer Protocol for Universities (Part 2)", *Byte*, vol. 9, no. 7, (July 1984), p. 143.

[Davies72] Davies, Donald W. "The Control of Congestion in Packet-Switching Networks", *IEEE Trans. on Communications*, vol. COM-20, no. 3, (June 1972), pp. 546-550.

[Day83] Day, John D. and Zimmermann, Hubert. "The OSI Reference Model", *Proc. of the IEEE*, vol. 71, no. 12, (December 1983), pp. 1334-1340.

[Demco88] Demco, John C. Personal communication, June 1988.

[Elie79] Elie, M. "Traffic Control by Latency in a Packet Switching", in *Flow Control in Computer Networks*, J.-L. Grangé and M. Gien, eds., North-Holland Publishing Co., Amsterdam, 1979, pp. 201-210.

[Feridun88] Feridun, M., Leib, M., Nodine, M., and Ong, J. "ANM: Automated Network Management System", *IEEE Network*, vol. 2, no. 2, (March 1988), pp. 13-19.

[Ferrari78] Ferrari, Domenico. *Computer Systems Performance Evaluation*, Prentice-Hall, 1978.

[Fishman73] Fishman, G. S. *Concepts and Methods in Discrete Event Digital Simulation*, Wiley, 1973.

[Forsdick80] Forsdick, Harry C. "AUTODIN II FTP", Report 4246, Bolt, Beranek and Newman, February 1980.

[Gambosi84] Gambosi, Giorgio, Bovet, Daniel P., and Menascé, Daniel A. "A Detection and Removal of Deadlock in Store and Forward Communication Networks", *Performance of Computer-Communication Systems*, H. Rudin and W. Bux (eds.), North-Holland, 1984, pp. 219-229.

[Gelernter81] Gelernter, David. "A DAG-Based Algorithm for Prevention of Store-and-Forward Deadlock in Packet Networks", *IEEE Trans. on Computers*, vol. C-30, no. 10, (October 1981), pp. 709-715.

[Gerla80] Gerla, Mario, and Kleinrock, Leonard. "Flow Control: A Comparative Survey", *IEEE Trans. on Communications*, vol. COM-28, no. 4, (April 1980), pp. 553-574.

[Gerla81] Gerla, Mario. "Routing and Flow Control", in *Protocols and Techniques for Data Communication Networks*, Franklin F. Kuo (ed.), Prentice-Hall, 1981.

[Gerla88] Gerla, Mario, and Kleinrock, Leonard. "Congestion Control in Interconnected LANs", *IEEE Network*, vol. 2, no. 1, (January 1988), pp. 72-76.

[Giessler78] Giessler, A., Hänle, J., König, A., and Pade, E. "Free Buffer Allocation – An Investigation by Simulation", *Computer Networks*, vol. 2, 1978, pp. 191-208.

[Giessler81] Giessler, Alfred, Jägemann, Annemarie, Mäser, Ellen, and Hänle, Jürgen. "Flow Control Based on Buffer Classes", *IEEE Trans. on Communications*, vol. COM-29, no. 4, (April 1981), pp. 436-443.

[Gopal85] Gopal, Inder S. "Prevention of Store-and-Forward Deadlock in Computer Networks", *IEEE Trans. on Communications*, vol. COM-33, no. 12, (December 1985), pp. 1258-1264.

[Gordon78] Gordon, Geoffrey. *System Simulation*, Prentice-Hall, 1978.

[Günther81] Günther, Klaus D. "Prevention of Deadlocks in Packet-Switched Data Transport Systems", *IEEE Trans. on Communications*, vol. COM-29, no. 4, (April 1981), pp. 512-524.

[Hahne86] Hahne, Ellen L., and Gallager, Robert G. "Round Robin Scheduling for Fair Flow Control in Data Communication Networks", *Proc. 6th Inter. Conf. on Comp. Comm.*, June 1986, pp. 103-107.

[Herrmann76] Herrmann, Jeff. "Flow Control in the ARPA Network", *Computer Networks*, vol. 1, 1976, pp. 65-76.

[Holt72] Holt, Richard C. "Some Deadlock Properties of Computer Systems", *Computing Surveys*, vol. 4, no. 3, (September 1972), pp. 179-196.

[Howard73] Howard, J. H., Jr. "Mixed Solutions for the Deadlock Problem", *Commun. ACM*, vol. 16, no. 7, (July 1973), pp. 427-430.

[Irland78] Irland, Marek I. "Buffer Management in a Packet Switch", *IEEE Trans. on Communications*, vol. COM-26, no. 3, (March 1978), pp. 328-337.

[Isloor80] Isloor, Sreekaanth S., and Marsland, T. Anthony. "The Deadlock Problem: An Overview", *IEEE Computer*, vol. 13, no. 9, (September 1980), pp. 58-78.

[ISO86] ISO. "Information Processing Systems – OSI – File Transfer, Access and Management", ISO/TC97/SC21/WG5, DIS 8571, July 7, 1986.

[Jaffe80] Jaffe, J. M. "A Decentralized, 'Optimal' Multiple-User, Flow Control Algorithm", *Proc. 5th International Conf. on Comp. Comm.*, October 1980, pp. 839-844.

[Joy83] Joy, W. N., Cooper, E., and Fabry, R. S. "Unix Programmer's Manual 4.2BSD", Computer Systems Research Group, EECS, University of California at Berkeley, 1983.

[Kahn72] Kahn, Robert E., and Crowther, William R. "Flow Control in a Resource-Sharing Computer Network", *IEEE Trans. on Communications*, vol. COM-20, no. 3, (June 1972), pp. 539-546.

[Kamoun80a] Kamoun, Farouk, and Kleinrock, Leonard. "Analysis of Shared Finite Storage in a Computer Network Node Environment Under General Traffic Conditions", *IEEE Trans. on Communications*, vol. COM-28, no. 7, (July 1980), pp. 992-1003.

[Kamoun80b] Kamoun, Farouk, Belguith, A., and Grange, J.L. "Congestion Control with a Buffer Management Strategy Based on Traffic Priorities", *Proc. 5th International Conference on Computer Communication*, 1980, pp. 845-850.

[Kamoun81] Kamoun, Farouk. "A Drop and Throttle Flow Control Policy for Computer Networks", *IEEE Trans. on Communications*, vol. COM-29, no. 4, (April 1981), pp. 444-452.

[Katevenis87] Katevenis, Manolis G. H. "Fast Switching and Fair Control of Congested Flow in Broadband Networks", *IEEE J. Select. Areas in Comm.*, vol. SAC-5, no. 8, (October 1987), pp. 1315-1326.

[Kermani79] Kermani, Parviz, and Kleinrock, Leonard. "Virtual Cut-Through: A New Computer Communication Switching Technique", *Computer Networks*, vol. 3, 1979, pp. 267-286.

[King86] King, Peter J. B., and Shacham, Nachum. "Queueing Models for Buffers with Dial-Up Servers", *Proc. IEEE INFOCOM '86*, 1986, pp. 616-625.

[Kingston86] Kingston, Douglas P., III. "MMDF II: A Technical Review", 4.3BSD contributed software, Computer Systems Research Group, EECS, University of California at Berkeley, 1986.

[Kleinrock80] Kleinrock, L., and Tseng, C. W. "Flow Control Based on Limiting Permit Generation Rates", *Proc. 5th Inter. Conf. on Comp. Comm.*, 1980, pp. 785-790.

[Klerer88] Klerer, S. Mark. "The OSI Management Architecture: an Overview", *IEEE Network*, vol. 2, no. 2, (March 1988), pp. 20-29.

[Konorski86] Konorski, Jerzy. "Store-and-Forward Deadlock Prevention in Packet Networks: A Circulating Token Approach and Performance Considerations", *IEEE International Conf. on Communications '86*, vol. 1, June 1986, pp. 119-123.

[Koorland85a] Koorland, Neil. "A Message-Based Remote Database Access Facility", M.Sc. Thesis, University of British Columbia, Dept. of Computer Science, August 1985.

[Koorland85b] Koorland, Neil, Gilmore, P. C., Vuong, S. T. "Providing a Remote Database Access Facility for the EAN Messaging System", *Computer and Information Science Association Conference*, June 1985, pp. 201-205.

[Lam79] Lam, Simon S., and Reiser, Martin. "Congestion Control of Store-and-Forward Networks by Input Buffer Limits – An Analysis", *IEEE Trans. on Communications*, vol. COM-27, no. 1, (January 1979), pp. 127-133.

[Lam80] Lam, Simon S., and Lien, Y. C. Luke. "An Experimental Study of the Congestion Control of Packet Communication Networks", *Proc. 5th International Conf. on Comp. Comm.*, 1980, pp. 791-796.

[Lam81] Lam, Simon S., and Lien, Y. C. Luke. "Congestion Control of Packet Communication Networks by Input Buffer Limits – A Simulation Study", *IEEE Trans. on Computers*, vol. C-30, no. 10, (October 1981), pp. 733- 742.

[McQuillan79] McQuillan, J. M. "Interactions Between Routing and Congestion Control in Computer Networks", in *Flow Control in Computer Networks*, J.-L. Grangé and M. Gien, eds., North-Holland Publishing Co., Amsterdam, 1979, pp. 63-75.

[Merlin80] Merlin, Philip M., and Schweitzer, Paul J. "Deadlock Avoidance in Store-and-Forward Networks, Parts I and II",*IEEE Trans. on Communications*, vol. COM-28, no. 3, (March 1980), pp. 345-360.

[Mills87] Mills, David L., and Braun, Hans-Werner."The NSFNET Backbone Network", *Computer Communications Review, Special Issue - Frontiers in Computer Communication Technology*, vol. 17, no. 5, 1987, pp. 191-196.

[Mitchell84] Mitchell, Don P., and Merritt, Michael J. "A Distributed Algorithm for Deadlock Detection and Resolution", *Proc. 3rd ACM Symp. on Principles of Distributed Computing*, 1984, pp. 282-284.

[Nagle84] Nagle, John B. "Congestion Control in IP/TCP Internetworks", *Computer Communications Review*, vol. 14, no. 4, (October 1984), pp 11-17.

[Nagle87] Nagle, John B. "On Packet Switches with Infinite Storage", *IEEE Trans. on Communications*, vol. COM-35, no. 4, (April 1987), pp 435-438.

[Neufeld86] Neufeld, G., Demco, J., Hilpert, B., and Sample, R. "EAN: An X.400 Message System", *Computer Message Systems - 85*, R. Uhlig (ed.), North-Holland, 1986, pp. 3-15.

[Nowitz78] Nowitz, D. A., and Lesk, M. E. "A Dial-Up Network of UNIX Systems", *Unix Programmer's Manual*, Seventh Edition, Volume 2B, Bell Laboratories, January 1979.

[Nowitz86] Nowitz, D. A., and Green, Ross. "Installation and Operation of UUCP - 4.3BSD Edition", *Unix System Manager's Manual*, SMM:9, Computer Systems Research Group, EECS, University of California at Berkeley, May 1986.

[Partridge88] Partridge, Craig, and Trewitt, Glenn "The High-Level Entity Management System (HEMS)", *IEEE Network*, vol. 2, no. 2, (March 1988), pp. 37-42.

[Postel81a] Postel, Jonathan B., Sunshine, Carl A., and Cohen, Danny. "The ARPA Internet Protocol", *Computer Networks*, vol. 5, 1981, pp. 261-271.

[Postel81b] Postel, Jonathan B. "RFC 792: Internet Control Message Protocol – DARPA Internet Program Protocol Specification", USC/Information Sciences Institute, September 1981.

[Postel82] Postel, Jonathan B. "Simple Mail Transfer Protocol", RFC 821, USC/Information Sciences Institute, August 1982.

[Pouzin73] Pouzin, Louis. "Presentation and Major Design Aspects of the CYCLADES Network", *Proc. 3rd Data Communication Symposium*, 1973, pp. 80-87.

[Pouzin81] Pouzin, Louis. "Methods, Tools, and Observations on Flow Control in Packet-Switched Data Networks", *IEEE Trans. on Communications*, vol. COM-29, no. 4, (April 1981), pp. 413-426.

[Pritsker69] Pritsker, A. Alan B. and Kiviat, Philip J. *Simulation with GASP II – A Fortran Based Simulation Language*, Prentice-Hall, 1969.

[Quarterman86] Quarterman, John S., and Hoskins, Josiah C. "Notable Computer Networks", *Commun. ACM*, vol. 29, no. 10, (October 1986), pp. 932-971.

[Rahnema88] Rahnema, Moe. "Smart Trunk Scheduling Strategies for Future Integrated Services Packet-Switched Networks", *IEEE Communications Magazine*, vol. 26, no. 2, (February 1988), pp. 43-50.

[Raubold76] Raubold, E., and Hänle, J. "A Method of Deadlock-free Resource Allocation and Flow Control in Packet Networks", *Proc. 3rd International Conference on Computer Communication*, 1976, pp. 483-487.

[Redell83] Redell, David D., and White, James E. "Interconnecting Electronic Mail Systems", *IEEE Computer*, vol. 16, no. 9, (September 1983), pp. 55-63.

[Rudin76] Rudin H. "An Introduction to Flow Control", ICCC, 1976.

[Sadowski84] Sadowski, Edward R. "The Efficacy of a Store-and-Forward File Transfer System", M.Sc. Thesis, University of British Columbia, Dept. of Computer Science, April 1984.

[Schwartz79] Schwartz, M., and Saad, S. "Analysis of Congestion Control Techniques in Computer Communication Networks", in *Flow Control in Computer Networks*, J.-L. Grangé and M. Gien, eds., North-Holland Publishing Co., Amsterdam, 1979, pp. 113-130.

[Shoch79] Shoch, John F. "Packet Fragmentation in Inter-Network Protocols", *Computer Networks*, vol. 3, 1979, pp. 3-8.

[Sunshine77] Sunshine, Carl A. "Interconnection of Computer Networks", *Computer Networks*, vol. 1, 1977, pp. 177-195.

[Tanenbaum81] Tanenbaum, Andrew S. "Computer Networks", Prentice-Hall, 1981.

[Teng83] Teng, A.Y., Yao, J., Gopinath, B., and Sabnani, K. "A File Transfer System for Scheduling File Transfers in the Bell Labs Network", *Proc. IEEE INFOCOM '83*, 1983, pp. 279-287.

[Thareja82] Thareja, Ashok K., Tripathi, Satish K., and Upton, Richard A. "On Updating Buffer Allocation", *ACM Computer Network Performance Symposium, Performance Evaluation Review*, vol. 11, no. 1, (Spring 1982), pp. 101-110.

[Thareja83a] Thareja, A. K., and Agrawala, A. K. "Impact of Buffer Allocation Policies on Delays in Message Switching Networks", *Proc. IEEE INFOCOM '83*, pp. 436-442.

[Thareja83b] Thareja, A. K., and Agrawala, A. K. "Characterization of an Optimal Delayed Resolution Policy", *Proc. ACM Sigmetrics Conf. on Measurement and Modeling of Computer Systems, Performance Evaluation Review*, August 1983, pp. 257-265.

[Thareja84a] Thareja, Ashok K., and Agrawala, Ashok K. "On the Design of Optimal Policy for Sharing Finite Buffers", *IEEE Trans. on Communications*, vol. COM-32, no. 6, (June 1984), pp. 737-740.

[Thareja84b] Thareja, Ashok K., and Tripathi, Satish K. "Buffer Sharing in Dynamic Load Environment", *Proc. IEEE INFOCOM '84*, pp. 369-379.

[Thomas85] Thomas, Robert H., Forsdick, Harry C., Crowley, Terrence R., Schaaf, Richard W., Tomlinson, Raymond S., and Travers, Virginia M. "Diamond: A Multimedia Message System Built on a Distributed Architecture", *IEEE Computer*, vol. 18, no. 12, (December 1985), pp. 65-78.

[Tipper88] Tipper, D., and Sundareshan, M. K. "Adaptive Policies for Optimal Buffer Management in Dynamic Load Environments", *Proc. IEEE INFOCOM '88*, pp. 535-544.

[Toueg79] Toueg, Sam, and Ullman, Jeffrey D. "Deadlock-Free Packet Switching Networks", *Proc. 11th ACM Symposium on the Theory of Computing*, 1979, pp. 89-98.

[Toueg80] Toueg, Sam. "Deadlock- and Livelock-Free Packet Switching Networks", *Proc. 12th ACM Symposium on the Theory of Computing*, 1980, pp. 94-99.

[Wimmer84] Wimmer, Wolfgang. "Using Barrier Graphs for Deadlock Prevention in Communication Networks", *IEEE Trans. on Communications*, vol. COM-32, no. 8, (August 1984), pp. 897-901.

[Wong82] Wong, J. W., Sauvé, Jacques P., and Field, James A. "A Study of Fairness in Packet-Switching Networks", *IEEE Trans. on Communications*, vol. COM-30, no. 2, (February 1982), pp. 346-353. Also appears as Technical Report E-Report E-90, Computer Communications Networks Group, University of Waterloo, June 1980.

[Wunderlich80] Wunderlich, E. F., Kaufman, L., and Gopinath, B. "The Control of Store and Forward Congestion in Packet Switched Networks", *Proc. 5th Inter. Conf. on Comp. Comm.*, 1980, pp. 851-856.

[Yum84] Yum, T. P., and Dou, C. "Buffer Allocation Strategies with Blocking Requirements", *Performance Evaluation*, vol. 4, 1984, pp. 285-295.

[Zhang87] Zhang, L. "Designing a New Architecture for Packet Switching Communication Networks", *IEEE Communications Magazine*, vol. 25, no. 9, (September 1987), pp. 5-12.

[Zwaenepoel85] Zwaenepoel, Willy. "Protocols for Large Data Transfers Over Local Networks", *Proc. of the Ninth Data Communications Symposium*, September 1985, pp. 22-32.

Glossary

AU	Access unit
CCITT	Comité Consultatif Internationale de Télégraphique et Téléphonique
CP	Complete partitioning
CQL	Channel queue limit
CS	Complete sharing
DR	Delayed resolution
DTFC	Drop and throttle flow control
EAN	An X.400-based message handling system developed at the University of British Columbia
FIFO	First-in, first-out
FTAM	File transfer, access, and management
FTP	File transfer protocol
ISO	International standards organization
MHE	Message handling environment
MHS	Message handling system
MMDF	Multi-channel memo distribution facility
MS	Message store

MTA	Message transfer agent
MTAE	Message transfer agent entity
MTL	Message transfer layer
MTS	Message transfer system
O/R Address	Originator/recipient address
O/R Name	Originator/recipient name
OSI	Open systems interconnection
RBSAD	Recipient buffer space allocation delay
RTS	Reliable transfer server
S/F	Store and forward
SDMXQ	Sharing with discrimination and maximum queue length
SMA	Sharing with a minimum allocation
SMQMA	Sharing with a maximum queue and minimum allocation
SMTP	Simple mail transfer protocol
SMXQ	Sharing with maximum queue lengths
UA	User agent
UAE	User agent entity
UAL	User agent layer
UUCP	Unix-to-unix copy
WORM	Write-once, read-many memory